AUERBACH

DYNAMIC SOFTWARE DEVELOPMENT

Managing Projects in Flux

TIMOTHY D. WELLS

<Half-Title Page>

<Title Page>

Library of Congress Cataloging-in-Publication Data

Wells, Timothy D.
Dynamic software development : managing projects in flux / Timothy D. Wells.
p. cm.
Includes bibliographical references and index.
ISBN 0-8493-1292-2 (alk. paper)
1. Computer software--Development--Management. I. Title.

QA76.76.D47 W47 2002
005.1′068--dc21

2002027956

No claim to original U.S. Government works
International Standard Book Number 0-8493-1292-2
Library of Congress Card Number 2002027956
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

Contents

1 Defining the Goal... Or Visualizing the Ideal 1
Skills and Success 1
Knowledge Management 4
Manager's Nightmare 6
Information as the Manager's Tool 6
  The Product's State 7
  The Element's State 7
  The Task's State 8
Trust What You Know, Not What You Are Told 8
Applying Dynamic Management 10
Chapter 1 Case Study Excerpt 12

2 Defining Work... Or What's Really Happening in the Trenches 19
A Day in the Life of a Developer 21
  8:00 a.m. 21
  8:30 a.m. 22
  9:00 a.m. 22
  11:30 a.m. 22
  1:00 p.m. 23
Relating Management to Work 24
A Unit of Work = Chunk of Information 26
Defining Tasks 28
Projecting Size 30
Work and Corroboration 33
Applying Dynamic Management 34
Chapter 2 Case Study Excerpt 36

3 Planning Progress... Or What You Don't Know Can Hurt You 41
Information Structure 41
Real Building Blocks 42
Reward Complete Thinking 49
Real Uncertainty 50
Watch the Result, Not the Process 51
Applying Dynamic Management 53
Chapter 3 Case Study Excerpt 54

4 Managing Developers... Or Dance with the One Who Brought You 61
No Management? No Documentation? 61
Diverse Skill Set 62
Combining Work and Learning 66
The Team of One 66
Multi-Team Efforts 67

Motivating by Rewarding Consistent Work 69
Applying Dynamic Management 70
Chapter 4 Case Study Excerpt 71

5 Monitoring Productivity... Or Getting Better All the Time 77
Measuring Work Done 77
Measuring What Has Changed 80
Determining Work's Cost 84
Demanding Enhanced Value 84
Applying Dynamic Management 86
Chapter 5 Case Study Excerpt 87

6 Strategic Framework... Or Metadesign Integrity 91
The Importance of System Architecture 91
Technology Decisions 94
Mapping Architecture to Elements 96
Architecture's Dark Side 97
Applying Dynamic Management 98
Chapter 6 Case Study Excerpt 99

7 Constructive Development Environment... Or Making Work Flow 103
Conflict within the Environment 103
Seven Components of a Development Environment 105
Sources of Conflict 106
Assessing the Environment 109
Ongoing Assessment 110
Applying Dynamic Management 111
Chapter 7 Case Study Excerpt 112

8 Managing Managers... Or I'm OK, but the Rest of Them? 117
Dealing with Expectations 117
The Delegate Channel 120
The Collaborate Channel 121
The Service Channel 121
Selling Enhanced Value 123
Applying Dynamic Management 124
Chapter 8 Case Study Excerpt 126

9 Funding and Economic Return... Or Paying the Way 131
Funding as Risk Containment 131
Funding the Perpetual Effort 135
Paying for Asset Development 137
Applying Dynamic Management 139
Chapter 9 Case Study Excerpt 140

10 Leadership by Consensus... Or if You're Going My Way 147
Decision Councils 147
Leadership through Criticism 149
Need for Responsibility 150
Applying Dynamic Management 151
Chapter 10 Case Study Excerpt 151

Postscript 157

Appendices

A Distorted Reality... Or Why Phased Management Is Appealing 159
What Is So Bad about the Waterfall? 159
Why Long Lead-Times? 159
Phased Management Leads to Specialization of Skills 160
Phase-Based Management Leads to Distortion of the Real Objective 160
Phased-Based Management Leads to Lost Information Assets 161

B Where to Begin... Or Getting Started with Dynamic Management 163
Selecting Comrades 163
Orchestrating Pilot Projects 164
Define Your Building Blocks 164
Define Your Repository 164
Measure Productivity 165
What You Can Lose 165

C Capability Maturity Model and Dynamic Software Management 167

D Dynamic Management Information Model 185
Information Model Definitions 185

E Glossary 189

F Reading List 195

G DSM Case Study 197
Overview 197
- Organization 197
- Environment 199
- Time: Twelve Years Ago 201
- Time: Eleven Years Ago 205
- Time: Ten Years Ago 206
- Time: Nine Years Ago 212
- Time: Eight Years Ago 215
- Time: Seven Years Ago 226
- Time: Six Years Ago 228
- Time: Five Years Ago 230
- Time: Four Years Ago 232
- Time: Two Years Ago 234
- Time: One Year Ago 237

Conclusion 238

Index 241

Introduction

I have been thinking about writing this book since I missed my first deadline. I was the new graduate working on his first big project. I worked obediently through the phases of a well-defined (...or at least voluminous) project methodology. All the forms were prepared with great care. I did not know the exact purpose of each form, but each was so meticulously laid out that I assumed it had to be important. First came "Feasibility Study," then "Functional Requirements," then "Preliminary Design." The project repository began to fill with impressive looking three-ring binders. After a while, all the binders began to look alike, and the forms housed within began to blur together — each with its proper heading and section numbers — each with its three approving signatures duly dated.

Then the big push came! The project was behind schedule. We worked nights and weekends. We were so busy, I did not even notice that we never referenced the beautiful three-ring binders we had spent so much time preparing. We had no time. We had to deliver the system.

> It seemed strange to me that the project was "suddenly" behind schedule.

I survived the period of firings and resignations. I was the junior member of the team. I could not be blamed. I was the naïve, wide-eyed programmer. I wondered about the logic behind working so hard on requirement specifications, only to ignore them when the schedule became tight. Where was the rationale? What was the motivation? I kept thinking my managers knew what they were doing. In time, I thought, I would be enlightened with their wisdom.

I later become a software management consultant for Yourdon, Inc. I wondered at the three-ring binders built by my clients. I asked if they were fulfilling some useful purpose. Usually, the answer was a qualified "Yes." I wondered as the three-ring binders were replaced with text databases, CASE tools, project management software, and knowledge repositories. Still, if the project found itself behind schedule (a common occurrence), the information in those electronic three-ring binders was referenced less and less, as if the value of that information had suddenly dropped. Or, perhaps, the code suddenly became more valuable. What was the motivation? What was the rationale? What was the use?

Enlightenment gained definition during those years. It was obvious that the form of the information was not relevant. Computer technology was more flexible than wood fiber technology, but the effect was the same. The information was not useful in monitoring the state of the project. Somehow, the information in these volumes is not perceived as valuable when the parameters of the project change. The knowledge gained by the developers during the course of the project was useful insofar as the developers could remember it, but the project repository could, by no stretch of the imagination, be called an asset.

Now I am an independent consultant and a university professor studying the same phenomena I observed when I was a student, a developer, and a manager. I offer this text as an alternative to traditional project management. I define an information-based approach to software development and enhancement where the objectives are to gather information and add value, rather than to complete tasks. Work to be performed is determined by the information needed rather than the next step in the methodology.

This book is built around a model of dynamic software development. The domain in which software development managers work is a dynamic, ever-changing environment driven by three significant forces:

1. *Demands placed on system development* to meet corporate goals, strategic direction, and operational needs
2. *Managers' need for information* about the state of the products under development
3. *Elements of technology* used to create software products

As the enterprise's management defines new needs and set direction, system management defines tasks to investigate how the system of elements might be enhanced. The task's definition identifies strands of related elements into which the new requirement will be integrated. System management assigns the task to a development team based on the developer's skills, knowledge, and availability. The developers organize available technology into a set of elements that are integrated into the software product.

The development manager uses information about the product's elements to manage the development effort. These elements are the software code, requirements, test cases, designs, plans, task definitions, and other pieces of information recorded in a repository. The contents of the repository reflect what is known about the software product and the way it serves the corporation.

The cycle continues. Advancements in technology stimulate better methods of building software solutions. The development manager monitors changes to the repository and guides the development effort. Enhancements to the software product spur requests for added functionality. The changing business environment generates new needs and changes in strategic direction.

The dynamic management techniques defined in this text support all methods of development, but apply most constructively to extreme programming, adaptive and aspect-oriented programming. The ideas were formed over countless attempts to devise management techniques compatible with earlier methods such as Rapid Prototyping, radically concurrent development, and iterative development. All these methods yield significant benefits to the development process. At the same time, they are dissonant with traditional project management techniques. While developers welcomed the methods, project managers and directors greeted the new developments with skepticism.

This text defines the principles, practices, skills, and techniques needed to manage a dynamic development environment. Dynamic development recognizes that there is no beginning and no end to the software development process. There is no distinction between *maintenance* and *new development*. Development effort is perpetual, continuous, and involves infinite combinations of varying features, shifting requirements, and changing technology.

Each chapter ends with a section to help you apply the techniques to your specific organization. I invite you to work through the text. Evaluate the logic of the approach and the relevance of the examples. You will find ideas that will make an immediate difference in your organization. Perhaps you would be willing to contribute your experience and expertise to the emerging techniques of "dynamic software management." I invite you to visit and participate in our Web site at http://luminguild.com/dynamic.

Timothy D. Wells
Pittsford, New York
March 2002

Case Study Illustration

Excerpts of a case study are woven throughout the text. A section in each chapter presents a narrative describing relevant events in the history of a composite organization — DSM International. While the organization is not real, the case study is true.

All characters are realistic. I have observed all the ideas, feelings, actions, and reactions depicted in the narrative during my 30 years of developing, managing, consulting, and teaching. You will recognize situations and events because every organization faces the same dilemmas and decisions. Every organization deals with similar challenges. Every organization responds to these challenges uniquely.

I am not specific about the nature of DSM International. I believe the dynamics and problems illustrated in the narrative are common across corporate, nonprofit, and professional organizations. As you read the narrative, you should fill in details from your own experience. The people represented are identified only by titles rather than names. I am sure you will see your colleagues, your superiors, and your developers in the narrative.

Topics discussed in each chapter of the book are illustrated in the case study excerpts at the end of each chapter. You may find it useful to read through the case study. The case study is presented in chronological order in Appendix G. Then as you read the body of the text, the referenced segments of the case study will be familiar to you.

We learn by exchanging stories. I am sure as you read the text you will be reminded of important and illustrative events in your professional life. You are encouraged to share your stories in the discussion at the author's Web site (http://luminguild.com/dynamic).

Applying Dynamic Management

Each chapter ends with suggestion on how to apply and implement dynamic management in your organization. These suggestions are set apart by a horizontal bar. Many of the ideas in this segment assume you have some form of computer-mediated communication tool or electronic conferencing system in use in your organization. This type of utility allows for messages to be posted in threaded, asynchronous discussions accessible through your network.

An electronic conference can be an extremely effective aid to business process improvement:

- Ideas can be presented in a convenient, nonthreatening manner.
- Responses can be posted and reviewed by a wide audience without having to schedule meetings or distribute memos.
- Consensus can be achieved quickly with wider participation.

Many of the recommendations at the end of each chapter are in the form of topic definitions and open-ended questions you and your colleagues will find useful in evaluating the ideas presented in this text and planning effective methods of adapting them to your environment.

Format Conventions

Each chapter begins with a comment or aphorism.

> Setting the stage… summarizing the chapter.

Summaries and important points are set apart with bold type:

> **Scan the book for these comments to see a summary of important assertions.**

I could not resist making asides and impious remarks based on my own experience and emotional response. I also use this format to include quotes and aphorisms that complement the subject.

> You are free to ignore these comments, but I believe most readers will appreciate the sentiments.

Chapter 1

Defining the Goal...

Or Visualizing the Ideal

There is no end to software development. Managers must define the goal not as finishing, but as the act of managing well.

—Timothy Wells

Skills and Success

A vision of the ideal assists in making sense of an often-chaotic situation. The mental image acts as a template. When reality deviates from the template, you move quickly to bend the situation back toward the ideal.

During a management seminar I was conducting on the West Coast, I asked the participants to list the traits of a successful development effort. I got the following list.

- No one is surprised (all expectations are met).
- The users are delighted.
- After working with the system for six months, the users are still delighted.
- All the developers feel good about their work.
- After working with the system for six months, the developers still feel good about their work.
- Every change request is quickly and successfully addressed.
- Upper management feels they got more than their money's worth.
- The users know they got their money's worth.

- The users and upper management do not have to wait a year to know they are getting their money's worth.
- I get a big raise, a corner office, and I do not have to manage software development ever again.

These are the characteristics of an ideal development effort (with the possible exception of the last point). These qualities, to some degree, must be present from the very beginning and continue as the development effort expands and grows. We do ourselves a favor to watch for these traits. If you do not see them, it is time to take some restorative action.

No one is surprised (all expectations are met).

It is a common and comic situation. You sit down at a review meeting and run through a demonstration of the latest version. You see the indescribable but familiar look of the users' faces — that strange combination of puzzlement, anger, and exasperation. Their expression speaks volumes: "You idiot. That might have been what we asked for, but you should have known that's not what we want!"

Key skill of the development manager: managing expectations.

The users are delighted.... And after working with the system for six months, the users are still delighted.

Users need solutions to problems and real enhancement to their business environment. You cannot determine that these needs have been met at the time a new version is released. Many products have an attractive interface that can delight us, regardless of the functionality behind it. Humans tend to judge books by their covers and software is no different. The useful evaluation comes after the gloss becomes familiar and the product is integrated into the work environment. The real test is the durability of delight. If the user is still singing your praises after the system becomes comfortable, your only problem will be maintaining your humility. There is nothing more rewarding than seeing your software solution being used effectively. But have no fear. Delight is an extremely rare condition.

Key skill of the development manager: knowing and acting upon real needs.

All the developers feel good about their work.
After working with the system for six months, the developers still feel good about their work.

Development efforts have many audiences (e.g., users, managers, developers, suppliers, auditors, and regulators). Keeping users delighted is of utmost importance and that cannot be done without talented, motivated developers. There is nothing more frustrating than working for months on end on a failing system. There is nothing more motivating than working on a system that is important, people know it is important, and people have confidence that it is working because of one's efforts.

Key skill of the development manager: knowing and capitalizing on real talent and teamwork.

Every change request is quickly and successfully addressed.

This characteristic of success is central to the rest of the text. To address a new demand, managers and developers must have access to all the defined knowledge of the product. If that knowledge has been lost (a key employee has left) or corrupted (previous design changes were not recorded), our ability to respond to new expectations is seriously eroded. A manager must ensure the knowledge repository — that is, everything we know about the product — is accurate and available.

Key skill of the development manager: developing a knowledge repository, not just a product.

Upper management feel they got more than their money's worth.
The users know they got their money's worth.
The users and upper management do not have to wait a year to know they are getting their money's worth.

The key economic reality is that software development creates assets (or liabilities) in the form of corporate information and the ability to manage and utilize it. Development managers must make it clear that the resource invested in software systems is a good deal. You have a degree of discretion over the financial horizons we define. The successful manager provides frequent and consistent evidence that the resource being applied to information system development is money well spent.

Key skill of the development manager: managing the knowledge repository to maximize return on investment.

I get a big raise, a corner office, and I do not have to manage software development ever again.

This last point is clearly a personal ambition. In my years as manager, consultant, and teacher, I have noticed an interesting phenomenon: those who are good at managing the development of effective software solutions like what they do and are not inclined to move "up" to the corner office; and those who are not successful work toward the corner office and end up managing the successful development managers (a clear example of the Peter Principle). This is not always a destructive situation. If I am propelled up the ranks by encouraging the success of my employees, I am doing good work for myself and the corporation. If I am not successful at managing a development effort, but I create an environment where my people can, then I am successful.

Key skill of the development manager: climbing the corporate ladder by making it possible for others to succeed.

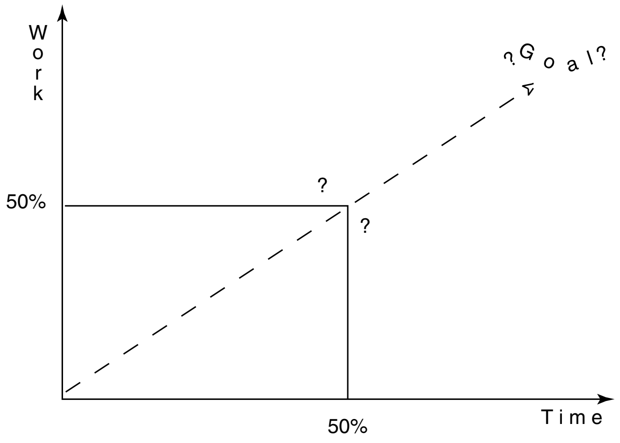


Exhibit 1 Unrealistic View of Development

This is the development manager's lot: watching, taking action, monitoring the effect, taking action, defining goals, taking action, gaining consensus, taking action, redirecting resources, taking action, redefining goals, and taking more action.

Knowledge Management

It is tempting to think of managing development as the process of guiding a team from a point in time when zero work is accomplished, to a point in time when all required work is done and the users delight in the results (see Exhibit 1). With this image, we could plan a neat linear path where 50 percent of the work will be complete when 50 percent of the time allotted for the effort has been spent. (See Appendix A for a detailed discussion of the distortions caused by this linear perspective.)

The reality is that the work is never done. You cannot know the extent of the required work. The conditions that will delight the user today are different from those that will delight the user tomorrow. The goal keeps changing — always toward more work. You do not manage work; you manage the knowledge produced from effective work.

Managing software development is an exercise in knowledge management.

A constructive image is to think of software development management as the ongoing activity of directing knowledge acquisition. You and your developers are gathering and creating knowledge about your business enterprise and the way software is applied to its improvement. The fact that the goal seems to change and shift is only natural. As you learn more about business and software solutions, you identify other needs and opportunities. The goal changes because the process is never complete.

Your job is to direct the process of gathering unstructured, inaccurate, contradictory, and intangible information and organizing it into a coherent, comprehensible, verifiable, tangible product. It would be convenient if the information we needed flowed to us in a nice steady, predictable stream. But information is dynamic, ever changing, and multi-dimensional in nature. You cannot hope to know everything at once and you cannot expect the information you know today to be valid tomorrow. To successfully manage software development, you have to become comfortable with uncertainty.

> I never hope to embrace the whole, but merely to give in each separate fragment of work, the feeling of the whole as I go on.
>
> **—Henry Miller**

I was having dinner with a client. He was a development manager for a midwestern bank. He said he had succeeded in putting to bed the latest enhancements to a deposit system. With his upcoming vacation on his mind, it was important to have the development effort in a steady state before he left. He wanted to be able to concentrate on the important issues of rest and relaxation. I thought the phrases "putting to bed" and "steady state" were descriptive. "The deposit system is one of the most active systems you manage." "We have 400 change requests every year for that system." "With needs changing at such a pace, how can you possibly put it to bed? A system like that is never in a steady state." "Well, the way I figure it, if we know what we don't know, it's steady."

Knowing that we do not know something is a real achievement in software development. Managing a development effort is perhaps the most information intensive of business activities. I track information from developers, users, senior managers, suppliers, competitors, and peers. I record tasks, timetables, wants, needs, demands, and results. I analyze the ramifications of proposed design, study the relationships in departmental data, ask what-if questions, and play through what-next scenarios. Losing track of the important information is often the norm. It has been described as a cartoon — ducking and dodging manuals, computers, documents, spreadsheets, and people swirling around your heads as you try to impose some order on the chaos.

My friend seemed to have learned how to effectively manage an inherently paradoxical environment. Successful managers develop a sense of what is important. They track the important items (even as the relative importance changes daily). They understand the warning signs of impending disaster and take small, timely actions to direct the effort toward a goal that cannot be achieved because it is always changing. During our dinner, my friend likened his job to herding cats. Some people seem to have developed the talent.

> Although this seems a paradox, all exact science is dominated by the idea of approximation.
>
> **—Bertrand Russell**

Manager's Nightmare

I once had a horrible recurring dream while working on a particularly ugly system development effort. The dream began in my office. My e-mail inbox was filling up faster than I could read the messages. Developers and users were flowing in my door, all talking at once, all demanding this and that.

Suddenly, I was no longer in my office. The scene was now some interdimensional void. It was quiet and eerie. I seemed to be floating in the middle of a holographic control panel. Gauge-like images floated around me. Most seemed to fluctuate a little but all seemed stable around the midpoint. I felt that the gauges reflected the status of various systems.

There were controls (or what I thought were controls) that responded to the proximity of my hand. Some seemed to respond to a passing thought or emotion.

As I tried to make sense out of the images, one of the gauge forms moved to the center of my field of vision. Its value was changing, as if demanding attention, but I did not know what to do. Other indicators soon began changing — as if a chain reaction was occurring — but I did not know the proper response! As each gauge's value grew more extreme, it moved closer to my face, demanding attention and competing with other dials.

I noticed some of the controls moving in harmony with the gauges. Desperately, I tried to influence the control I thought was associated with the most prominent gauge, moving my hands, searching for thoughts and feelings that would have some effect.

Something was working because the gauges and controls began to retreat to their original orbit around me. But soon, other gauge forms began changing, moving closer, growing more intense. Again I tried to determine the proper response. Was my hand movement doing anything, or were my thoughts actually moving the controls? Did the translucent lever or the amorphous wheel control the red gauge? Each time I seemed to gain a degree of control, a new and larger set of gauges began to misbehave. This scene would continue until I could wake myself up, panting and anxious.

There are likely to be many interpretations of the dream. I am sure there are deep psychological meanings. But most managers with whom I have worked have expressed anxiety not dissimilar from those of the dream. The successful manager develops the skill necessary to see the gauges changing before conditions become critical.

Information as the Manager's Tool

If the nightmare is a reasonable facsimile of a development environment, it begs the questions: What are the analogous gauges and controls? What does the manager monitor and what actions does she take?

- The gauges are the state of the product(s) and the state of the tasks that create the product's elements.
- The controls are the definitions of the products, the elements, and the tasks.

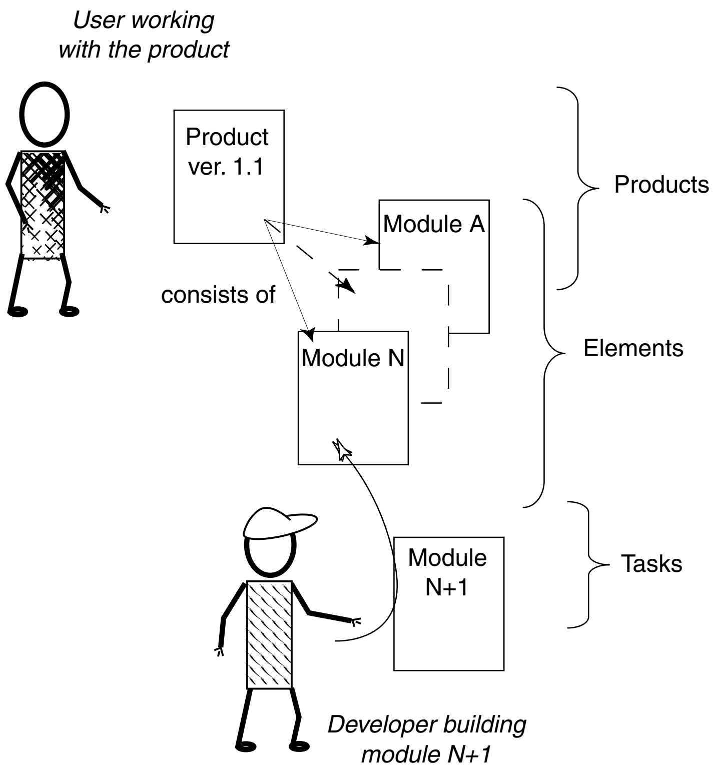


Exhibit 2 Development Repository

A product (e.g., a system or cohesive portion of a system) consists of elements (see Exhibit 2). There are many types of elements that define what we know about a product. A good example of an element is a module of code (C++ class, Java applet, or set of functions from a legacy system).

The Product's State

The product is part of the user's environment. If the product is helping the user succeed, the product is a success. With a successful product, users and upper managers continue to back the software development effort. If the product is perceived as a liability to users and their effort to advance the corporation, no amount of high-tech wizardry will convince them they are getting their money's worth.

The Element's State

A set of information elements (e.g., code, designs, requirements, test suites) makes up the product. If the set of elements fits together well, the state of the product is stable. If the set of elements does not fit together well (e.g., there are errors in the interface between modules), the product will be unstable, you will not have a pleasant vacation, and you will not get the corner office.

The Task's State

As new requirements for the product are identified, your developers work on additional modules and change existing ones to create the next version. If the task is defined and measurable, the work has a good chance of adding value to the repository of elements, keeping the product stable and contributing to the user's success. If the task is unrealistic or vague, the task will not be worth the investment in time and resource, the product becomes unstable, and the user and upper management begin to talk about your replacement.

To gather information, the manager watches the status of the products being produced — much like a broker watches the activity of an industry's stocks. Development managers take action based on this information, coupled with knowledge of available resources and known expectations.

The action takes the form of changing definitions:

- If the product's contribution to the enterprise's operation begins to lessen (due to changing requirements and new demands), the development manager might direct the development team to revise the product's definition by adding new functions, changing scope, or integrating new technology.
- The development manager watches the status of the elements. If error rates begin going up, or the number of user complaints increases, the manager might change the definition of a task and assign developers to rewrite key elements.
- While monitoring the tasks being performed by the developers, the development manager might notice productivity decreasing or error rates increasing in one development team. She might modify the composition of the team by regrouping the developers or changing the definition of the skills required for the task by scheduling training.

The development manager must use the information being created in the development process like a control panel — monitoring, in real-time, the changing status of the many dimensions of the effort (see Exhibit 3).

There are dozens of gauges in the development repository. The development manager is interested in the number of elements of various types, number of relationships between elements, the number of elements in various states, the change in these numbers over time, and various ratios — just to name a few. Later chapters discuss and define these and other gauges useful in managing the development effort.

Trust What You Know, Not What You Are Told

Humans tend to communicate in a way that bends and stretches reality. A manager asking a developer if a particular module will be ready for testing by the end of the week will likely be told yes (unless the task is demonstrably impossible). Humans tend to speak what is expected rather than what is. For

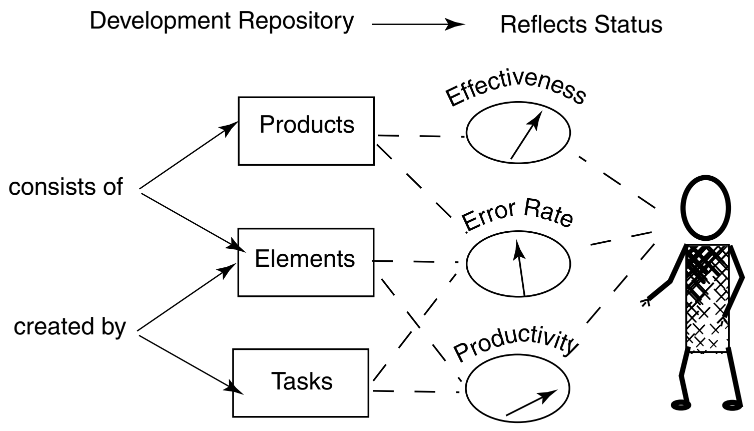


Exhibit 3 Manager's Gauges

this reason, a development manager must develop the habit of trusting the reality of the development repository, rather than the projections of developers or the desires of other managers. If the repository shows that 15 modules have been added in the past week, then it is likely that 15 modules will be added next week (all things being equal). The fact that the user wants 30 modules completed is irrelevant to the estimation process.

The development repository is not only an effective management tool for monitoring the development effort, but it is the only reliable tool for projecting the future (also known as estimating). The development repository contains the information reflecting the information gathered to bring the software products to their current state. In essence, it is a journal of actual changes that occur over time (see Exhibit 4). For example, one month ago there were eight elements in the repository. Today, there are 17 elements in the repository. If you project there are 30 additional elements to be built, you can expect the effort to be completed in approximately 3⅓ months working at a rate of nine elements each month.

This example is making many assumptions, including:

- All elements are of equal size.
- All the new elements are about as complex as the older elements.
- Older elements do not have to be changed as a result of adding new elements.
- The development staff does not change.
- The same tools are being used.
- The same architecture is being used.
- The user is involved to the same extent.
- The same quality is expected.

Later chapters address how to ensure these assumptions are valid or deal with situations where they are not.

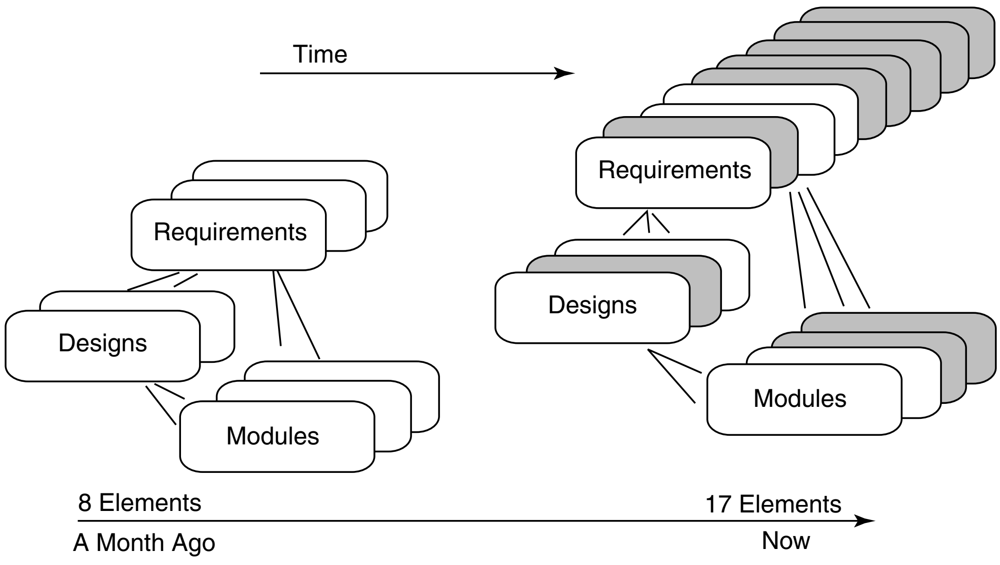


Exhibit 4 Repository Changing over Time

The information necessary to monitor the ongoing development campaign is in the development repository. It is the capital asset that fuels strategic planning. It is the object of development work. It reflects the development organization's contribution to the corporation. The essence of managing software development is the creation and effective use of this information asset.

Applying Dynamic Management

Invite project managers to an informal gathering. Start with a couple of colleagues. Give them a copy of this book and suggest getting together over lunch to discuss it.

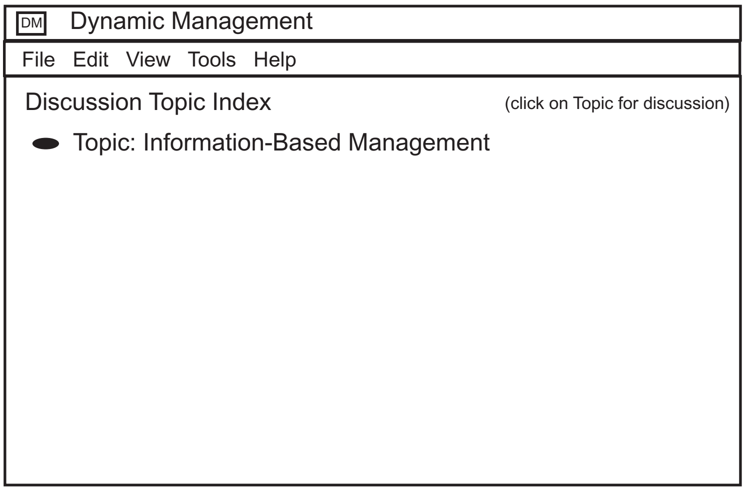
Before the first gathering, listen to your developers as they work on some enhancement task and take note of the types of information they reference. Take an inventory of the work products created by your developers and make a list of the types of information recorded in the work products. Try to identify associations between pieces of information.

The information and associations you identify are the knowledge repository of your development organization. Assume that your development organization has a convenient way of recording and accessing this information (see Exhibit 5).

Talk with your colleagues over lunch about the idea of *managing a development effort based only upon the changes in information in the repository.* Relate any experiences you have had about being surprised to discover the status of a development effort was significantly different from what your formal reports recorded. Discuss possible reasons for these discrepancies between what you thought and what turned out to be real.

Exhibit 5 Worksheet for Knowledge Repository

Element	*Description*	*Location*	*Association*
C++ Class		Project Directory/ Source Code Control	Object Design /*some associated with table schema */
Object Design		Rational Directory	C++ Class
Entity Definition		Data Modeler Directory	Table Schema /*overlaps with Object Design?*/
Table Schema		DBMS Metadata	Entity Definition
Use Case			Object Design
Test Suite			
etc.			



Discussion Topic Index (click on Topic for discussion)

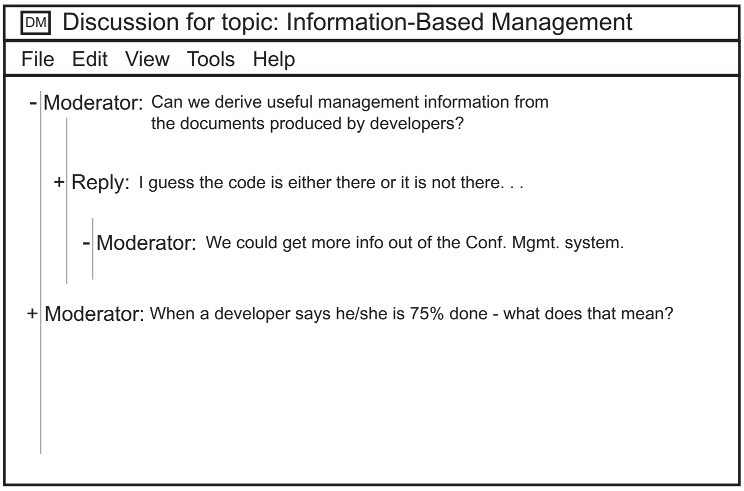
● Topic: Information-Based Management

Exhibit 6 Online Discussion Index

After your meeting, set up a discussion conference on network and invite your colleagues to continue the discussion online. You can start by posting a topic outlining your goals and objectives (see Exhibit 6).

Online discussion requires that you initiate the debate with some open-ended question. Post a set of questions that you feel relate to current issues important to your organization (see Exhibit 7). For example:

- Can we derive useful management information from the documents produced by developers?
- When a developer says he is 75 percent done, what does that mean?
- Would you need status reports if you could monitor the development repository accurately?
- If you could monitor the quality of the information in the repository, would you be as concerned about process?


DM
Discussion for topic: Information-Based Management
File Edit View Tools Help

- Moderator: Can we derive useful management information from the documents produced by developers?
  + Reply: I guess the code is either there or it is not there. . .
    - Moderator: We could get more info out of the Conf. Mgmt. system.
+ Moderator: When a developer says he/she is 75% done - what does that mean?

Exhibit 7 Beginning the Dialogue

Visit the author's discussion conference at http://luminguild.com/dynamic/ to see what other managers have to say.

Chapter 1 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

The first major development effort after the creation of the DSM Development Method (DSM-DM) was a product to support DSM clients. The marketing group was reporting that the advice and support from DSM was useful, but some form of computer-aided system would be useful. Some major clients reported that other firms were promising software support within the year and Marketing was concerned that these clients would opt for competing firms if DSM did not introduce computer-aided support.

The CEO met with the Director of Applications Development and they agreed to make the DSM Strategic Integration Support System (DSM-SISS) product a priority and that it was a great opportunity to "do the job right" by strictly following the new DSM Development Method.

All the developers had received training in DSM-DM. The methodology has been used in part on several small efforts, although the development staff had never been obliged to use the methodology strictly as written. The Director, confident that the development staff could deliver the DSM-SISS within a year's time, quickly initiated a project plan, laying out the phases and reviews necessary to bring the project in on time (see Exhibit 8).

The plan did allow for overlap of the phases. There was no need for a formal feasibility study as everyone agreed it was the best (and necessary) business decision and none of the managers in Applications Development felt there was any real technical risk.

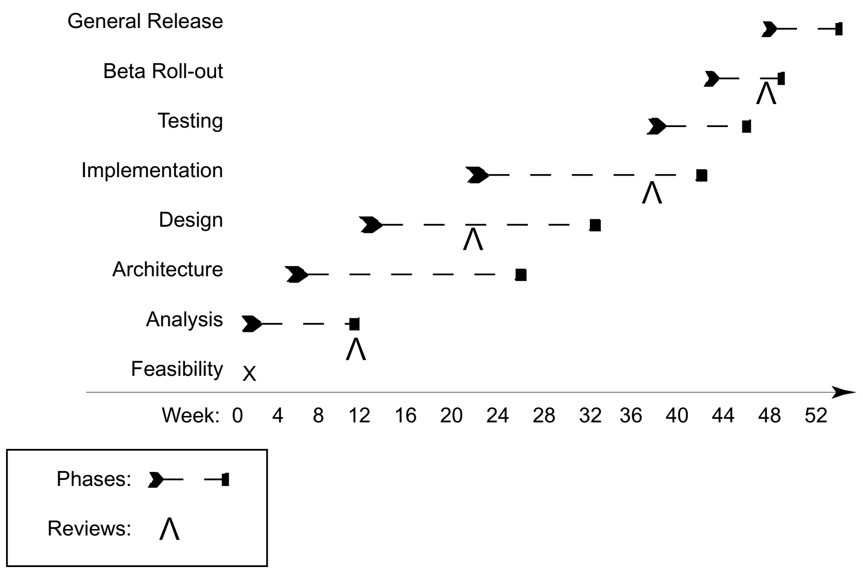


Exhibit 8 Overview: DSM-SISS Development Plan

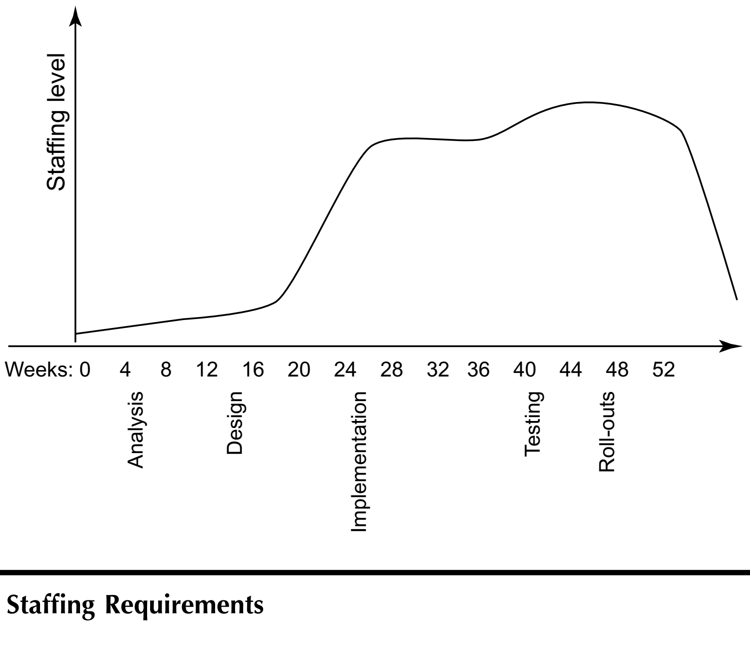


Exhibit 9 Staffing Requirements

Along with the schedule, the development plan outlined the staffing requirements for the project (see Exhibit 9). The first weeks of the project would require the services of three senior analysts. By the 12th week, three designers would be added. The project would be fully staffed by week 28 with additional testers expected around week 45. The majority of this complement could be reassigned to other projects by week 52.

Exhibit 10 E-Mail to Director Expressing Concern and His Response

To:	Director of Applications Development
From:	Sr. Analyst
Subject:	DSM-SISS Development Plan

I feel I must express some misgivings about the development plan we defined for the proposed SISS system. I can't help feeling we are creating our own reality. We think we know what the requirements of the system are, but we probably will not know for sure until week 12. If we discover by that point in time that the requirements for the system are much larger than we now expect, will we be obliged to work within the current time frame? It feels funny to be committing to a completion date when we are not sure of the amount of work we need to do.

The response:

To:	Sr. Analyst
From:	Director of Applications Development
Subject:	DSM-SISS Development Plan

Thanks for your message. I know that planning and estimation are not exact sciences. If we discover the product is much larger than we anticipate, we will make the necessary adjustments to the plan.

While the plan seemed consistent with the DSM-DM, one of the senior analysts sent an e-mail to the Director expressing some concern (see Exhibit 10).

The analysts started working the requirements documents. They got good support from the regional managers who were very excited about the prospect of adding the software support tool to their mix of products and services. They were already talking (informally) with clients about the new system. But every interview with a regional manager or client identified additional features and conflicting priorities.

Shortly before the first formal review, the analysts were going crazy trying pull together some coherent, consistent statements of system requirements but at the review meeting it was clear that the requirements were not close to being complete, the scope of the project was far greater than expected, and the frustration of the analysts was unmistakable. The executives at the meeting refused to consider extending the target date for the product. The VP of Marketing insisted that the product must be available on schedule or they would begin losing important clients to the competition.

After the review meeting, the Director of Applications Development sends out the e-mail shown in Exhibit 11.

At this point, the number of developers working on DSM-SISS increased as designers and programmers were assigned to the project. The development team took the memo (Exhibit 11) from the Director as a license to ignore aspects of the methodology they found inconvenient. After all, the goals had been clearly communicated: get something out the door by week 52 —

Exhibit 11 E-Mail after Review Meeting

To:	Development Team
From:	Director of Applications Development
Subject:	DSM-SISS Project

The project review of week 12 was not a great success. It is clear that we have a lot of work to do. Talking with many of the developers, I am confident that we can deliver a quality product in the time remaining. The analysts on the project have been directed to freeze the specification as it stands and to proceed with the Architecture and Design phases as defined in the project plan. Given our time constraint, we will look to streamline the tasks defined in the DSM-DM. Our next management review is scheduled for week 23. Let's pull together and deliver a product we and our clients can be proud of.

although the product is much larger than originally envisioned while the project plan remained unadjusted.

Managers were a little concerned as they heard the company's development methodology, DSM-DM, being referred to as "diz-dumb." But none of the managers publicly objected to the sarcasm because they knew they were working long hours and were pulling together as best they could to accomplish a difficult objective.

A few days before the scheduled week 23 review, the Director of Applications Development met with one of the regional managers. The Director pulled up the Event Definitions, Class Models, and Database Schemas for the system and was prepared to ask questions he had received from the development team. But before the Director could begin describing the various development models, the Regional Manager told the Director that several DMS clients had already received beta versions of similar products and that the features that seemed to be the most valuable were not even included in the original requirements statements. The Regional Manager said he wanted to discuss the new features at the review meeting and was wondering if the Director could prepare some estimates for what it would take to include them in DMS' product offering.

The Director, trying to do the best job he could, arranged to postpone the review meeting for one week and met with his managers and several of the senior analysts to prepare some response. They projected the new features would increase the size of the system by about 30 percent, revising the schedule to show the product rolling out about week 65.

When this revised plan was presented at the review meeting, the executives were not pleased. The CFO said she was uneasy about continuing the project. She quoted the total in salaries and overhead consumed by the project to date and wondered aloud what this money was buying. The VP of Marketing reasserted that the product was critical to the success of the company and that if more people were needed to get it done, then more resources should be allocated. The Director of Applications Development pulled out the Event Definitions, Class Models, and Database Schemas for the product to try to explain the status of the project, but most of the people in the room were unimpressed. The CFO's comment was, "These don't mean anything to me." The Director responded by pointing to the DSM-DM to support his position, but this did not carry much weight. The CEO saw that the meeting was going nowhere and ended it by saying he would be meeting with people individually to work out the next step.

The next week was a long series of meetings and hallway discussions. The next Friday, the CEO called the Director into his office. When the Director arrived, he saw that the VP of Marketing was already there. The Director was told that for some time now, the company had been looking into the possibility of buying out a small private company in the Midwest. The owner was retiring and was interested in selling his business. The Director recognized the company as one that had a nice local niche, but never competed with DSM outside the northern Midwest region.

The VP of Marketing reported that the Midwest company had a product that was exactly like the one DSM was building, and it was already in use by clients in that region. He was recommending that DSM buy out the Midwest company and market the product under DSM's name. The CFO had already reviewed the finances and her opinion was that the purchase would be good for DSM. The CEO turned to the Director and said he wanted him and one of his analysts to fly out and review the Midwest product and come back with a recommendation.

Before leaving, the Director told his development team about the move to buy a competitor's product and instructed them to stop pulling overtime for a few days and to turn their attention toward completing the documentation and cleaning up the work they had completed so far in preparation for whatever came next.

The Director and the analyst did not want to find a workable product at Midwest, but they did. They found a very different development environment and a software system that performed well with most of the needed features. Better yet (or worse, depending on your point of view), the Director and the analyst came to believe that the Midwest product was currently functional and that missing features could be added in subsequent releases. The Director reluctantly called the CEO and recommended that DSM proceed with the purchase and that Marketing can go ahead and start planning for the wider release of the Midwest product as DSM-SISS.

While the Director was at Midwest, he and his counterpart had several interesting (if not philosophical) discussions. Despite the circumstances, the two had a lot in common. One day, they were having lunch in the conference room and began sharing the impressions of the state of software development. The Director drew a graph on the whiteboard, with one axis labeled "type of work" and one labeled "time" (see Exhibit 12). He commented how bizarre it seemed to break up the y-axis by phase — feasibility, analysis, design, etc. He commented: "When I work on software, I am thinking about all these things at once … but one feature at a time."

As they talked, the Midwest manager drew a third axis to represent the features or functions of a proposed system. This made the Director even more

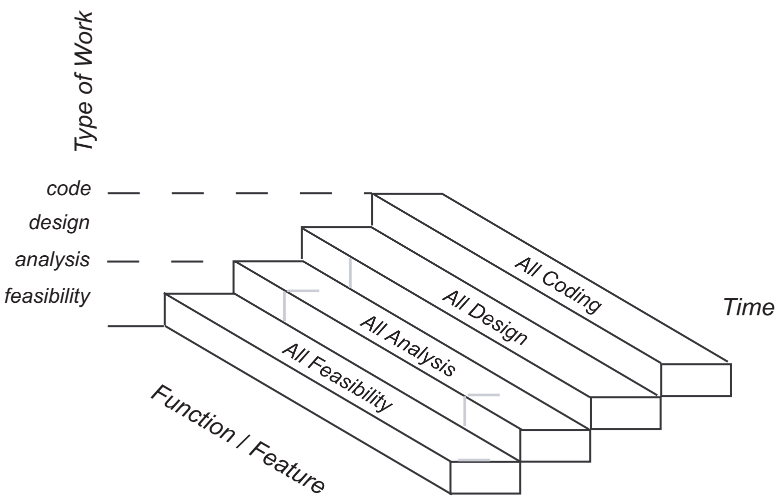


Exhibit 12 Steps

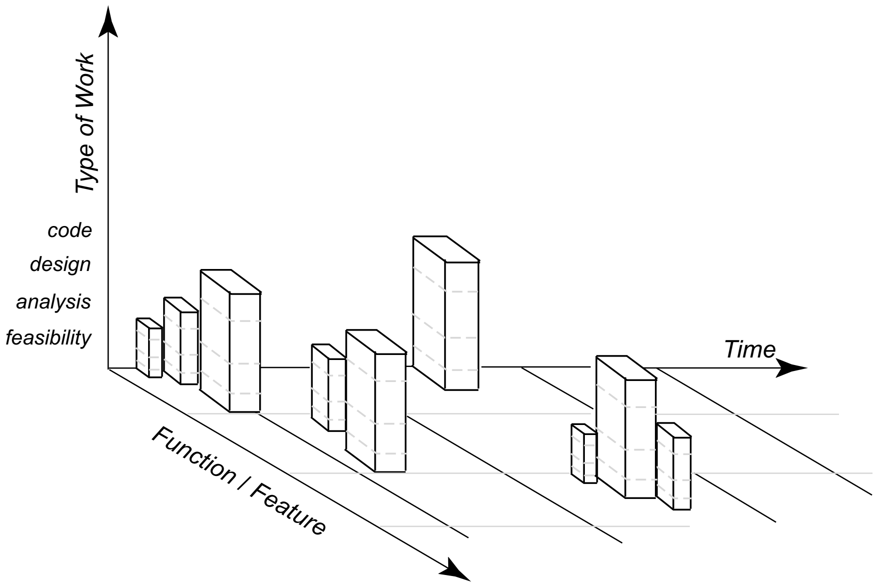


Exhibit 13 Clumps

irritated. "This drawing suggests we are trying to gather all information about feasibility in phase one, all information about requirements in phase two, etc. This is crazy! We can't be that sure about this information. By the time we get to design and coding, the lower steps have crumbled."

As the two finished lunch, the Director erased most of the drawing and began doodling. "If we could manage in a manner compatible with the way I work, we would recognize that software is built in clumps (see Exhibit 13). We build a cluster of functions — performing analysis, design, coding, and testing on them. At a later point in time, we build more software. At some point we may find a need to enhance the original work. At the same time, someone else is adding another feature — doing analysis, design, coding, and testing.

The Director's counterpart commented that that was pretty much how his group worked. "It has the advantage of always having a demonstrable system." The Director flashed back to the review meeting. He stood there, feeling rather stupid, with his database schema and class diagrams, unable to show that any real work had been done. The work was real but not in a form that could be appreciated by the decision makers.

They agreed to continue the discussion. The Director left Midwest with a sense that the two had a lot to learn from each other.

Chapter 2

Defining Work…

Or What's Really Happening in the Trenches

Manage what is real, not what is simply convenient.

—Timothy Wells

Management of any activity must be based on a definition of the desired results. The desired result is a product that adds value to the user and organized knowledge about the product, allowing you to enhance the product as the organization grows.

Chapter 1 introduced the idea of a development repository (see Exhibit 1) from which the development manager monitors product development. The information in the repository is valuable to the extent that it accurately reflects the reality of the development effort. Information that distorts the true status of the product, its elements, or the tasks creating the elements is a liability. Information that is accurate and easily accessible gives the development manager a distinct advantage in maintaining a successful effort.

Software products are made up of a variety of elements or chunks of information. During the development effort, tasks are defined and assigned to developers who then create a collection of related information. Most of the information created by developers is about the product's elements and their associations. Developers create code and associated designs. The code satisfies various requirements and is verified by building test cases.

All this information and all the needed associations are recorded somewhere across the project repositories, configuration management tools, computer-aided software engineering tools, project plans, files on individual developer's laptops, and in the memory of the programmer.

As a developer works, elements progress through various states. For example, the developer identifies the need for a unit of code. The code

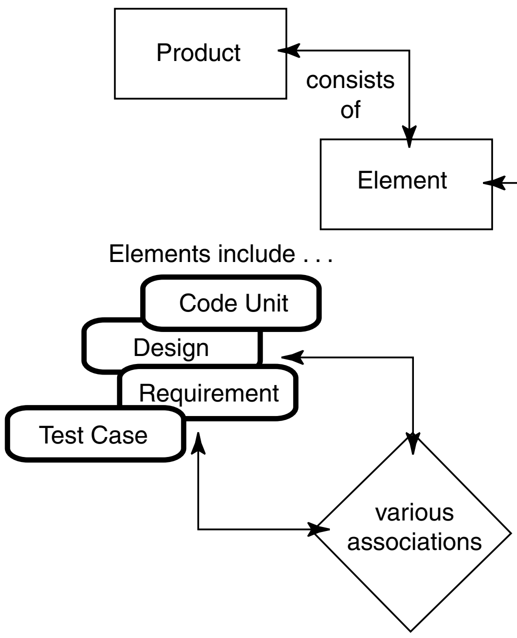


Exhibit 1 Development Repository

element is "declared." The developer writes the initial version of the code. Now the code element exists, but it is not reviewed. Another developer reviews the code and concurs that it is correct and meets relevant standards. The code unit now is "verified." Changes in the state of an element are artifacts of the developer's work.

Management of the development process can be viewed as the process of monitoring the repository and taking steps that move all elements — wherever they are recorded and whenever they were created — toward the verified state. A developer changing a class definition puts all associated elements in doubt. This doubt is the artifact of the developer's work. The class code and the designs in which the class is referenced may no longer be accurate. Someone has to perform work to review and possibly modify the associated elements. When all associated elements are reviewed and back, the repository is back in a steady state, the task is done, and the change has been successfully implemented. But if the doubt is ignored, the information about the product becomes less accurate and less able to support continued enhancement.

Managing the development effort means tapping the repository of development artifacts for the information needed to monitor and direct.

The goal is to move all elements in the repository to the "verified" state even as new elements are being added and existing ones are being changed.

I never did anything worth doing by accident; nor did any of my inventions come by accident; they came by work.

—Thomas Alva Edison

A Day in the Life of a Developer

Examining a developer's workday and noting the information used and created during the act of building software identifies the information needed for effective management.

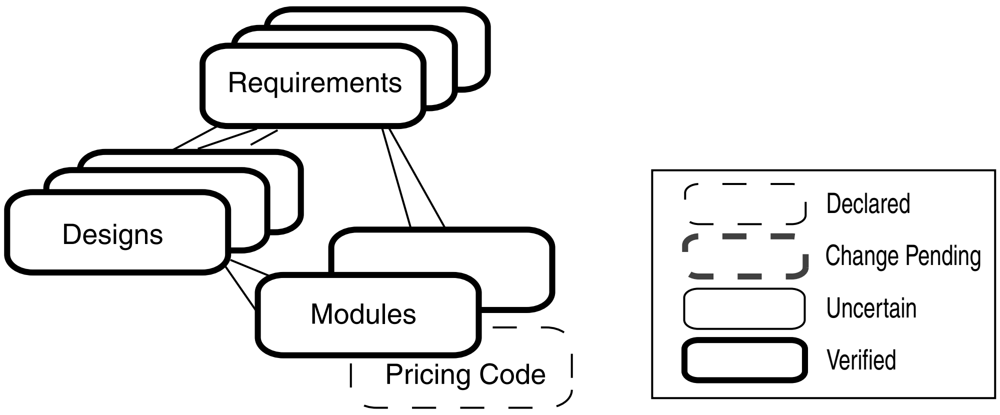


Exhibit 2 New Element

8:00 a.m.

A developer arrives at the manufacturing plant after a fitful night's rest. After parking and passing through security, he meets his manager in the coffee room. She is on her second cup, having arrived a half-hour earlier. The team is working on enhancements to the product pricing function in support of a new product line. As of this morning, the repository contains a handful of requirements, designs, and modules of code. The manager and her developer discuss the need for a particular piece of code they call the "Pricing Function" (see Exhibit 2). The function is essentially the same as code written for an older manufacturing system still in use.

Manager's information:

- The fact that a unit of code is needed implies that a certain amount of work must be performed.
- The repository reflects a new element in the "declared" state.

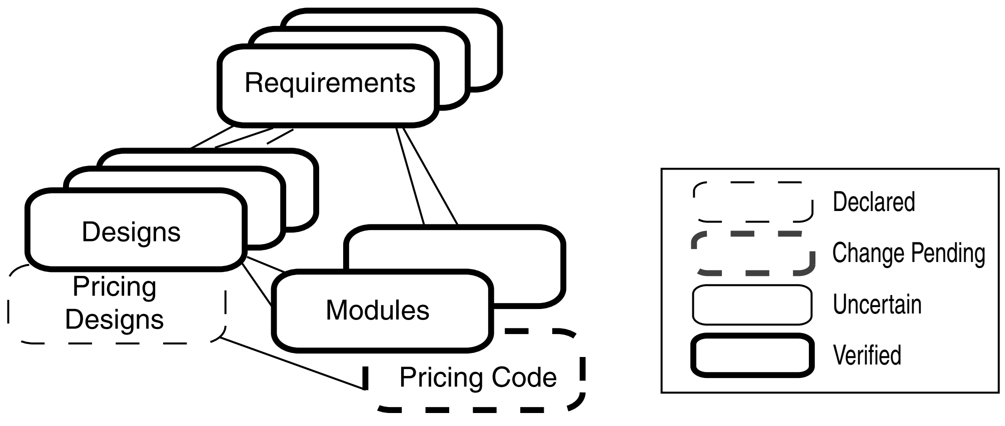


Exhibit 3 Add Associated Design

8:30 a.m.

The developer looks up the code used in the legacy system and determines it is "not bad." He copies the code into the repository. The organization's development standards require all code to have a corresponding design. Because the code is from an old application, the design is lost or was never created. For the knowledge in the repository to be complete, design information must be created, thus necessitating additional work (see Exhibit 3).

Manager's information:

- Now code exists for the "Pricing Function," however, the manager cannot be certain it is completely acceptable.
- By adding the code to the repository, the state of the repository reflects that some progress has been made and that a design is now needed.

9:00 a.m.

The manager walks into the developer's office and reminds him that all code must have an associated design before it is reviewed. The developer is not surprised. He has already begun reengineering the design to be sure he understands the old code and to plan changes needed for the new product pricing.

11:30 a.m.

Once the new design is complete and the pricing code has been updated, the repository reflects two "uncertain" elements. This set of related elements

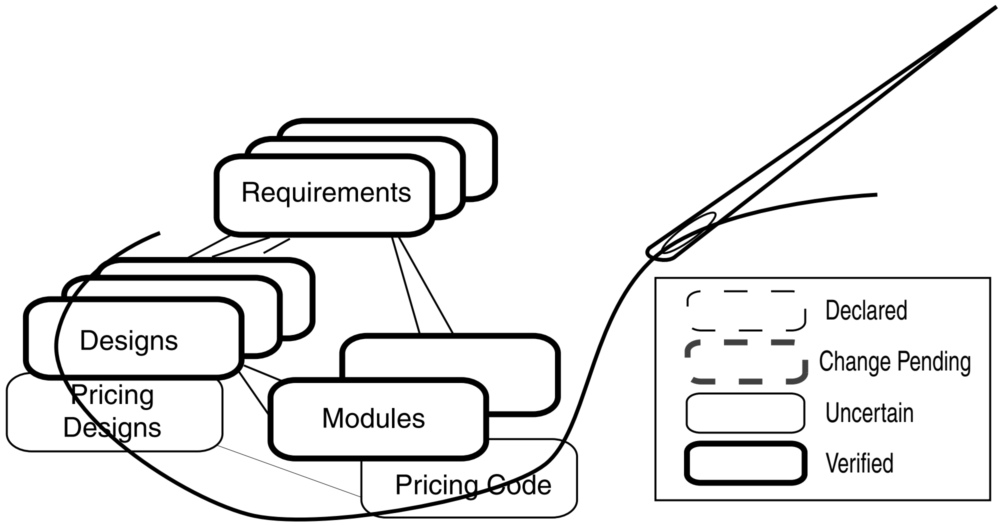


Exhibit 4 Strand of Elements

form a strand (see Exhibit 4). The strand is the subject of a quality review. Once the developer considers the strand complete, he submits it for review. Other developers will review it to ensure it is complete, understandable, correct, and ready to be included in the product. Later chapters address the review process in greater detail.

Manager's information:

- Now that the strand (Pricing Function code and design) have been created, reviewing the strand will move the product closer to the goal of having all elements created, reviewed, and ready to go.

1:00 p.m.

Two other developers get together and call up the design and code for the Pricing Function. They also have access to the company's standards policies, which they use as a guide to the inspection. They read the design and code, and come to an agreement that they are consistent and correct (see Exhibit 5).

Manager's information:

- Now the repository is in a consistent state.
- No additional work is implied by the state of the elements.
- It required the time and talent of three developers to move the repository from its original state to the one depicted in Exhibit 5.
- One developer found and modified a unit of code, and then created the corresponding design information.
- Two others reviewed and verified the new elements.

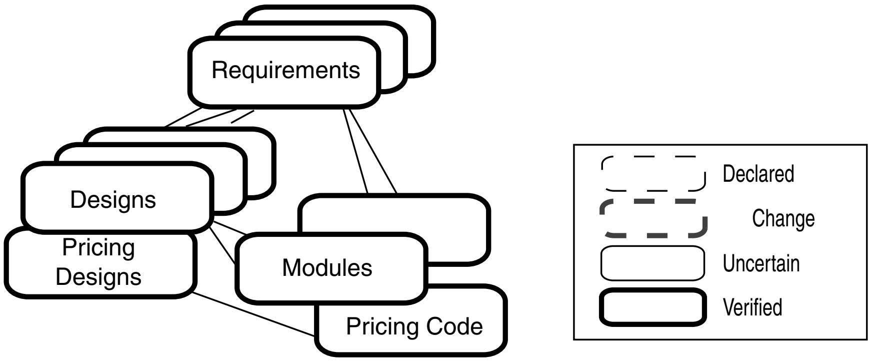


Exhibit 5 Repository in Steady State

Relating Management to Work

The Pricing Function scenario illustrates the creation of a cohesive strand of elements. The maintenance of this information is of utmost importance. Clear, accessible, and accurate information is important to your ability to use the repository for planning and tracking purposes. The organization of information is judged useful or not useful by the way that information contributes to future activities. If the information available to you makes it difficult to respond to future changes, the information's value is diminished. If the information is not transferable to other developers, the information's value depreciates when developers leave.

Knowledge without sense is double folly.

—Gracian

To illustrate, consider several alternatives to the idea of organizing the repository by strands. I have seen many software development efforts organized as if the developer owned portions of the product. Development departments are often organized around the individual developers (see Exhibit 6). One developer is assigned a set of tasks to be completed in relative independence. This approach seems to produce elements at a faster rate and is very appealing when an organization rewards individual effort over teamwork and cooperation.

This organization suffers as the complement of developers changes. As new developers join the team and others leave, knowledge of the elements deteriorates. Working individually, developers have little need to record their own knowledge thoroughly. Their individual memories serve them in their individual tasks. This organization hides the fact that the knowledge stored in the developer's memory is actually the information assets of the enterprise. Rather than recording the information in accessible formats, the organization has chosen to store the knowledge of elements and their associations in the memory of the developers.

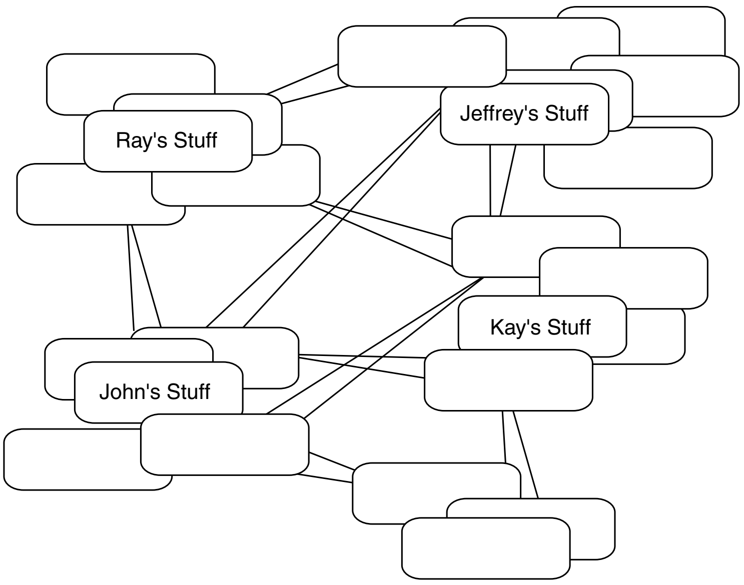


Exhibit 6 Repository Organized by Individual

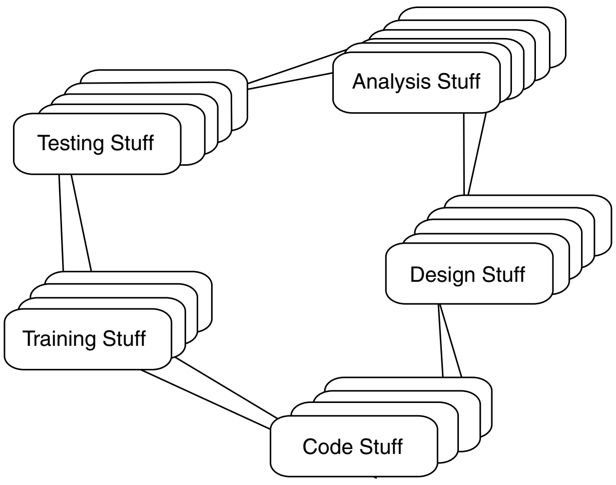


Exhibit 7 Repository Organized by Element Type

Another popular theme is to organize elements by type (see Exhibit 7). This follows the traditional phased approach. All the analysis work is done (meaning that all the requirements elements are created), followed by all the design work (meaning that all design elements are written). The design phase is followed by the coding phase (i.e., all the code elements are built). The elements in this organization are typically recorded by phase. All analysis material is recorded in documentation created during that phase. All code information is stored together with other information created during the coding phase.

This approach has the appearance of being efficient. We have all experienced the sense of satisfaction when the designs for a product are declared complete. But serious flaws emerge from the fact that associations between related elements are difficutl to maintain and recreate. The relationship between an analysis element and the design describing its implementation are recorded in two separate sets of documentation. The code that implements the design is often ill-described. When changes in the business environment require changes in the software systems, it is difficult to identify all the elements that might be affected because the knowledge necessary to evaluate the total effect is organized by the time frame in which it was built. Traceability has to be built in as an afterthought.

A problem that occurs for both alternatives just discussed is that important associations between elements are artificially separated. A developer writing code cannot not think about the design upon which the code is based. A user describing the need for a change in the product refers to an interface but is thinking about an underlying requirement. The change to the interface must be understood in relation to the requirement, its design, and the technology used to implement it. Developers reviewing a code can only evaluate syntax and adherence to standards if the design documents are inaccurate or unavailable. Code is good to the extent that it is consistent with the design and performs and delivers the functionality described in the requirement. A design is good if it effectively describes the implementation of a requirement and if an implementation can be produced from it.

At any given moment, the repository must provide a complete set of information needed to address the demands from the users and enterprise management. When the user asks to change an interface, the repository must provide the interface specification, its design, related requirements, and test cases in order to assess the size and ramifications of the required change. The development manager must be able to pull on a strand attached to the interface specification and have all associated elements pop up (see Exhibit 8). Only with the complete set of information can reasonable decisions about the product's development be made.

Any organization of the product's repository that increases the cost of accessing needed information is sub-optimal. The more time it takes to collect all relevant information, the lower the probability all needed information will be found and the lower the probability the repository will be changed to reflect the proposed change.

A Unit of Work = Chunk of Information

Defining and directing work is a basic management activity. You are constantly giving directions to developers to do something. The question now is: exactly what is that "something" you are asking them to do? Given the way people naturally work and the goal of software development, the short answer is: add or change a particular element and make any other changes necessary to make it fit.

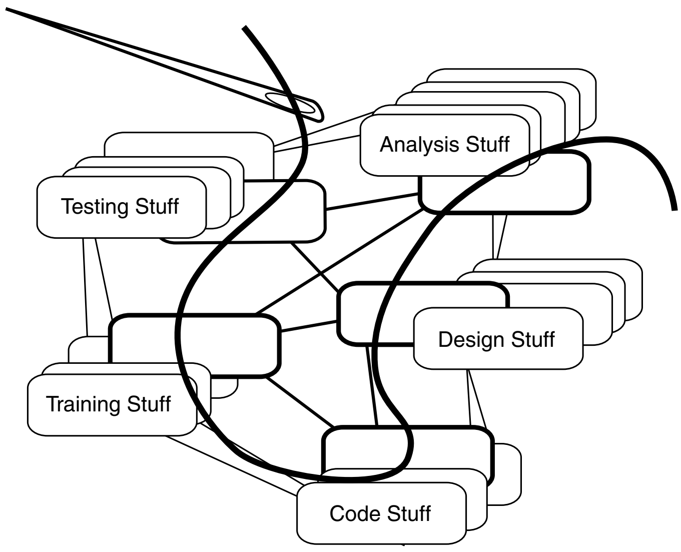


Exhibit 8 New Strand of Elements

Information is more useful if it is structured as frameworks of related items. We tend to learn more effectively when information is connected in relevant strands or chunks. We feel more intelligent if our knowledge is organized in a manner that allows us to recall relevant information in response to important stimuli. A manager is more effective if she can respond with an accurate project of resource needs when presented with a proposed enhancement request. A designer feels more confident about changes to a component design if he is sure he knows all related interfaces. A programmer is more accurate if he is confident that the data set definition is up-to-date.

Developments in artificial intelligence shed light on the nature of thought and work. Dennis Mercadal's *Dictionary of Artificial Intelligence* provides some useful definitions:

Chunk: information stored and retrieved as a single entity. The information is somehow related, although the different pieces of information in the chunk may be of different data types. Miller found that people learn most effectively by dealing with chunks of meaningful information. The chunks may be fairly large pieces of organized information. They are a collection of facts stored and retrieved as a single unit. In chess, a chunk may be the placement pattern of the entire chess-board. Research has indicated that one important difference between an advanced chess player and a beginner is the experience that allows the advanced player to have more chunks. This insight points out the importance of attempting to organize information in chunks. When we do this, we are employing more of a holistic approach rather than a strictly logical approach.

Chunking: The storing of information about a topic or object so that information is easily accessed. A process that takes place in learning in which information is abstracted in chunks. These chunks of information are stored, recalled, compared, and matched with patterns found in subsequent situations.

Previously, I defined the goal of software development as the creation of a repository of product knowledge where all elements were in a verified state. In an ideal world, every software requirement would be well understood, with cost-effective technology chosen to implement the user's needs. The implementation would be an accurate reflection of the requirements and consistent with standards. Rigorous test suites would be defined to ensure error-free implementation. A system with hundreds of strands would be easily understood because user, managers, and developers would be able to work one strand at a time, keeping the complexity of any task within manageable limits. Ramifications of proposed changes would be more visible, making planning more accurate and implementation more thorough. Changes to a product would result in changes to all related information, making the information in the repository as accurate after the change as it was before.

If you feel a twinge of cynicism, you are not alone. The fact that we have so little faith that our current management approach will produce useful information suggests a need for a different approach.

Clearly, the ideal will never exist. Change always involves risk. Risk invites error. However, we can move closer to the ideal by defining tasks that have as their primary goal the enhancement of the repository. We move farther away from the ideal if we address enhancement by changing only the code and data definitions while losing the association with other important elements of the product. The product is not only the code; the product is everything we know about the product and how it was built and how it is used.

The goal of software development is not so much to complete a task as to add value.

Defining Tasks

To build and maintain a useful repository, you must define work aimed at the goal. A task must identify an element to be created or modified, called the focus element. The focus element and elements associated with it make up the task's strand. The developers assigned to the task are authorized to modify the elements in the strand to ensure accurate and consistent information.

In the Pricing Function example earlier in this chapter, the focus element was the code unit. The code and its design formed the strand. The developer's task was to build and integrate the elements into the repository. To complete the task, the developer performed a set of actions or steps. In the scenario, the following development steps were taken:

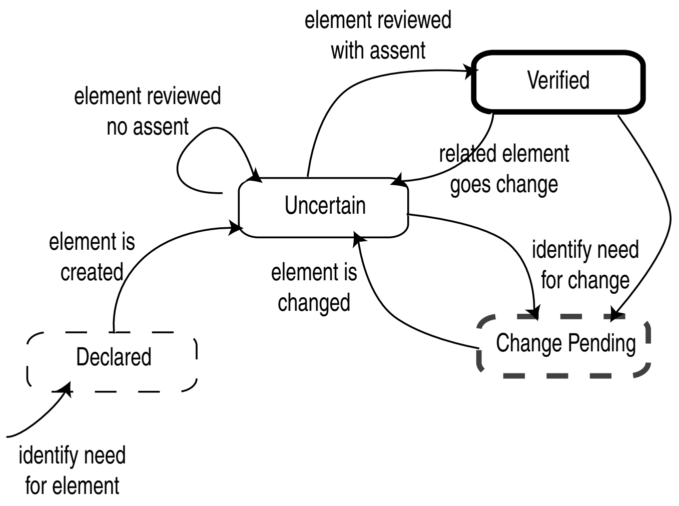

element reviewed with assent
Verified
element reviewed no assent
related element goes change
Uncertain
element is created
element is changed
identify need for change
Change Pending
Declared
identify need for element

Exhibit 9 Element State Model

1. Find and evaluate pricing function code from legacy system.
2. Reengineer design for old pricing function code.
3. Modify design and code for new product line.
4. Review design and code.

Action is the proper fruit of knowledge.

—Anonymous

When the development manager assigns a task to a developer, she identifies the focus element and the element strand. The developer then determines the development steps needed to complete the task. Less-experienced developers need more assistance in identifying the steps, while more seasoned developers proceed with little or no guidance. Chapter 4 addresses techniques for managing your development staff.

The objective is to add value to the repository by doing whatever is necessary to add required elements and ensure their "fit." The fact that the code for the Pricing Function existed but a design did not was an inconsistency. Resolving the inconsistency makes the repository more complete and easier to understand and therefore more valuable. Changing the design to reflect the requirements of the new product line makes the design inconsistent with the associated code, thus reducing the value of the repository's information. Updating the code to reflect the design brings the repository into a consistent and useful state. Having another developer review and verify the elements adds assurance that the design and code are correct, understandable, and usable.

The development steps taken during a task change the state of repository elements. Every element in the repository is moving through its life cycle as depicted in Exhibit 9. An element is identified. It is created. It is reviewed.

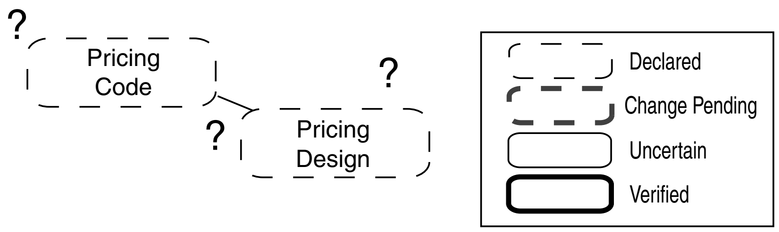


Exhibit 10 Strand for Task 1

Changes are identified. Problems are identified during the review. Corrections are made. The element is reviewed again.

A *task definition* consists of a focus element. All relevant associated elements constitute the *task strand*. By assigning a task to a developer, the manager sets the task's objectives: create or change the focus element and any element of the strand so that all elements are in a verified state.

Projecting Size

Most software developers have a sense of the size of their assigned tasks. That sense may be optimistic or lack accuracy, but it is derived from their guess of the number of "things" they do not know — but have to find out.

In the product pricing scenario discussed earlier, the developer knows code exists that is similar to the code needed from the current product (depicted in Exhibit 10). He does not know:

- If the old code will require rework
- If a design exists in the old documentation
- If the old design and the old code are consistent
- If the work he does today will be free of error and omission

The developer must perform work to discover the things he does not know:

- Study the old code and compare it with current requirements.
- Find the old design or recreate it.
- Compare the code to the design, identifying discrepancies.
- Have revised code and design reviewed by another developer.

To the extent that the repository definitions reflect the information needed to define the product, the repository content reflects the work that needs to be performed.

While one developer is working on the Pricing Function, another developer is working to fulfill a requirement to make a particular set of data available to intranet users. She believes she can create a class by extending and importing existing Java classes (Exhibit 11). She does not know:

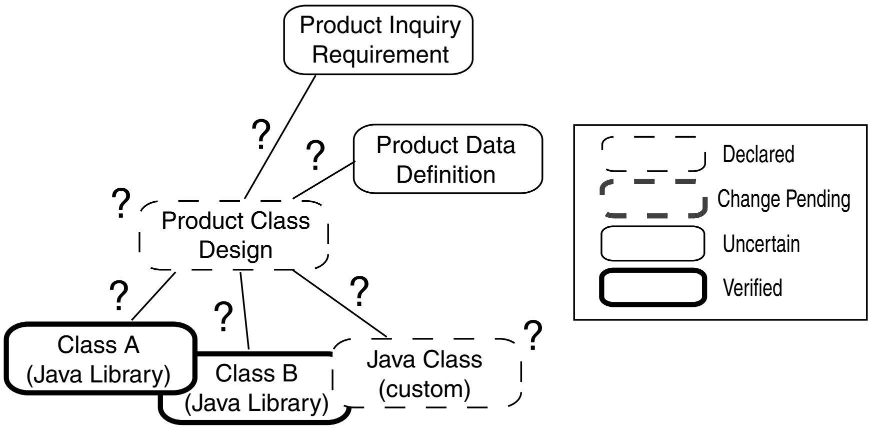


Exhibit 11 Strand for Task 2

- The definition of the class design
- If the new class design will fulfill requirements
- If a design succeeds in processing the data object
- If the existing Java classes (a and b) are appropriate for the design
- The syntax for the Java class
- If the design and the code will be consistent
- If the work she does today will be free of error and omission

The developer must perform work to find out what she does not know:

- Build the class design.
- Study the requirements.
- Study the data object definition.
- Evaluate existing Java classes A.
- Evaluate existing Java classes B.
- Write the syntax for the new Java class.
- Verify that the design and the code are consistent.
- Have the new code and design reviewed by another developer.

These job steps are clearly not independent. Most developers begin to build a design as they are studying requirements and thinking about code syntax while the design is still incomplete. It is natural for the developer's mind to jump from element to element as she works on a strand. This is how consistency is guaranteed. After the work is completed, the repository has been improved (i.e., it contains more information and the information is internally consistent). Conceptually, in addition to having two new elements, the question marks depicted in Exhibit 11 have been removed.

Although the eight activities were performed in parallel, eight pieces of work were performed. This is approximately twice as many as required for the first scenario depicted in Exhibit 10.

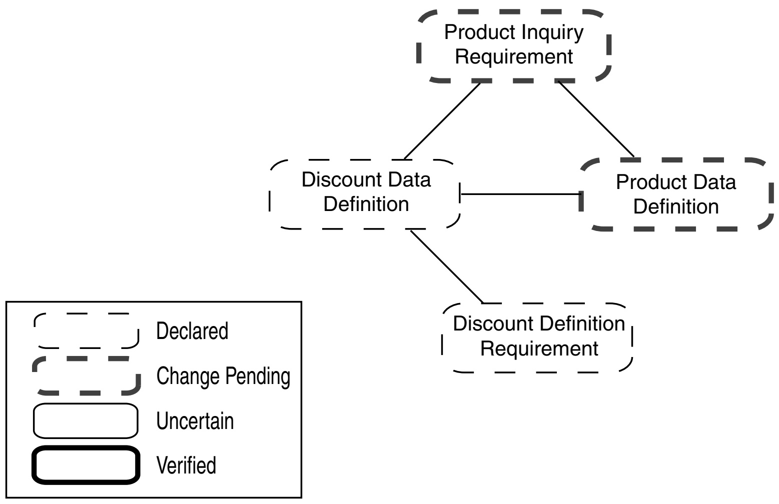


Exhibit 12 Strand for Task 3

The amount of work to be performed is proportional to the number of unknowns in the strand.

While the pricing function is being updated and the Java application is being built, a third developer is working to add a new data object definition and new requirement to the repository. The requirement being defined involves the definition of product discounts (Exhibit 12). The developer is on his way to discuss these with the product manager. The developer does not know:

- The actual content of the requirements
- The actual data element definitions
- If the requirements and the data definition will be consistent
- If the new data definition will affect the product inquiry requirement
- If the new data definition will affect or be affected by the product data definition

The developer must perform work to find out what he does not know:

- Interview the product manager and record the discount requirements.
- Interview the product manager and record the data defining a discount.
- Verify that the requirements are consistent with the data definition.
- Study the product inquiry requirement to determine if it must be altered.
- Study the product data definition to determine if it must be altered.

This scenario illustrates the perpetual relationship between completing some work and generating more work. If the developer determines that adding the discount elements to the repository necessitates changes in the associated elements, additional work is accrued. For example, if the original definition of the product included some discount data, adding a more thorough definition of discount as its own object implies the need to change the product object to remove the old discount attributes. Another example of accrued work is the product inquiry requirement that should be changed to display the discount information along with the product definition.

> Only when we know little do we know anything; doubt grows with knowledge.
>
> **—Wolfgang Von Goethe**

Determination of size can and should be based on actual work patterns. As knowledge workers, software developers are adding to the repository of known elements. The goal is to move all declared elements to a verified state. If a declared element is associated with ten other elements, it is likely to require twice the work of a declared element associated with only five elements; this is because each association implies pieces of knowledge that must be created or analyzed.

From the developer's perspective, the concept of a focus and a strand are helpful in defining what must be done. The focus and the strand also provide an answer to the question: How much work must be done?

A task's size is determined by three factors:

1. The number of elements and associations in the strand
2. The type of work to be done
3. The complexity of the elements

A complete determination of a task's size is based on the number of elements and associations in the strand, the type of work being performed, and the complexity of the element. In this chapter I have focused on the first of these factors. Chapter 3 (Planning Progress) and Chapter 5 (Monitoring Productivity) address the other two factors.

Work and Corroboration

Each of the examples in the previous section ended with a step to ensure that enhancements to the repository are free of error and omission. The work is not done until it is corroborated. Someone other than the author must read it and record the fact that they agree it is "up to snuff."

> **Information is not a valuable asset until it is accessible and can be correctly understood by any qualified reader. Information that is a secret (or unverified) is a liability.**

Many managers find it tempting to define the review or verification activity as a separate task. While there is an apparent advantage to being able to schedule reviews at later time, it also separates the activity that creates information from the activity that verifies the usefulness of the information.

As soon as the developer moves the elements depicted in Exhibit 10 to an "uncertain" state (e.g., the design and code are written), another developer should evaluate the work and either concur or disagree with the author's work. Referring back to the Element State Model (Exhibit 9), "reviewed with assent" means that the reviewer has read the work and certifies that it is understandable, correct, and complete. Reviewing "without assent" puts the element in a change pending state because the reviewer has identified what he believes to be an error or omission. Additional work must be performed to correct the error (or make it clear to the reviewer that no error or omission exists). This additional work is not part of a new task because the original task is not complete. The developers are still working toward the originally stated goal. The evidence that the goal has been reached is that all elements in the strand are in the verified state. Chapter 4 addresses the management of development teams in detail.

We work not only to produce but to give value to time.

—Eugene Delacroix

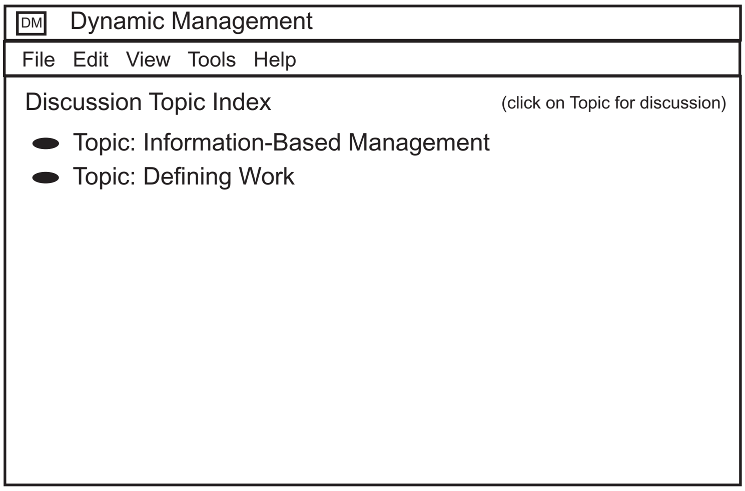
Development managers use this evidence to track progress and measure productivity. The verification and collaboration is necessary not only to ensure a quality product, but also to ensure that management information is accurate and complete.

Applying Dynamic Management

It is important that the definitions for elements and associations be derived from your development organization. There are no (and never will be) industry-wide standards for what elements and associations should be built and maintained in all development repositories. Applying dynamic management techniques to your organization requires element definitions based on your successful products. You and your developers create sets of useful information in order to create useful applications. Elements are definitions of that information.

The information you need to monitor and track a development effort should be a subset of the information used and created during the development process. The examples in this chapter imply that Java classes, class designs, business requirements statements, and data object definitions (e.g., database table definitions) are elements to be defined in the project repository.

This may be true for many development organizations but not for all. It is important that people in your development organization define the elements known to be important in the creation and enhancement of software applications.


DM Dynamic Management

File Edit View Tools Help

Discussion Topic Index (click on Topic for discussion)

- Topic: Information-Based Management
- Topic: Defining Work

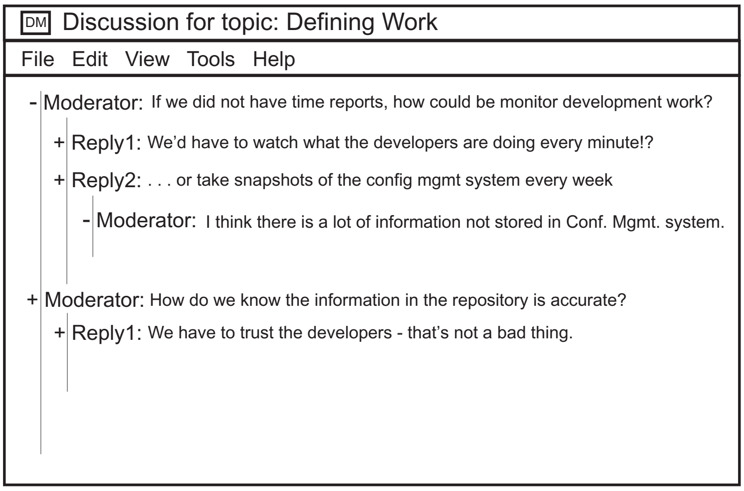
Exhibit 13 Online Discussion Index

At your next informal manager's meeting, identify an existing system built by your development department (see Exhibit 13). Based on your current understanding of your project archive, discuss the following questions:

- Would you feel comfortable saying to a new colleague that he will get an accurate understanding of the business requirements of a system by reading the information in the archive?
- Would you feel comfortable telling a new programmer that he will get an accurate understanding of the software design by reading the information in the archive?
- Discuss ways that the information in your development repositories can be kept accurate.

Assume you have a convenient way of querying all the work products produced by your developers. Also assume you have no direct information from your developers (i.e., your developers did not turn in time sheets and progress reports). The only source of information you have is their work products. Post the following questions to your online discussion group (see Exhibit 14):

- How would you measure the progress of the team?
- If one of your developers claimed he had performed Herculean amounts of work, but you were skeptical, how would you verify the developer's claim?
- If one of your developers looked like she was not working hard (e.g., she seemed to be spending a lot of time away from her desk — lots of time reading rather than pounding the keyboard), could you use the repository to measure her real contribution?


DM
Discussion for topic: Defining Work
File Edit View Tools Help

- Moderator: If we did not have time reports, how could be monitor development work?
  - + Reply1: We'd have to watch what the developers are doing every minute!?
  - + Reply2: . . . or take snapshots of the config mgmt system every week
    - - Moderator: I think there is a lot of information not stored in Conf. Mgmt. system.

+ Moderator: How do we know the information in the repository is accurate?
  - + Reply1: We have to trust the developers - that's not a bad thing.

Exhibit 14 Development Work Dialogue

Chapter 2 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

Before DSM adopted the DSM-DM methodology, work was done at CMM level 1. That is not to say that the work was bad. In fact, the development group had a good reputation throughout DSM for being fairly responsive and for producing systems that worked and were reasonably usable. When the development organization was smaller, it was workable to have each developer keep track of his or her own documentation. But as the group grew in size and the systems they were building grew in complexity, documentation became unworkable. Each developer's unique idiosyncrasies made it awkward and time-consuming to share information and build upon each other's work.

There were mixed feelings about working with the outside consulting firm in writing the DSM-DM methodology, but there was general consensus that it was a positive step. After all, everyone seemed to agree that a standardized set of phases in the development process was a good thing. Each phase would have a well-defined set of pre-conditions and expected results. The process would move smoothly from beginning to end. And everyone seemed to agree that there were (or should be) a definite beginning and a definite end to each development effort.

The DSM-DM worked well for the first year. There were no big projects, but lots of small ones. The Director of Applications Development allowed the developers to adapt and customize the methodology for each project so there were few complaints about redundant or unnecessary steps.

During that year, the development group adopted a new timesheet (see Exhibit 15) for reporting work spent on the various tasks defined in the DSM-DM. The report was easy enough. The Director and project managers would

Exhibit 15 Sample Time Report

Timesheet week of: 5/14

Date	*Hours*	*Task Code*	*Description*
5/14	4	C12-4	Writing code for task 12-4
5/14	2	C12-9	Updating data dictionary
5/14	2	Misc.	Correcting requirements errors
5/15	8	C12-4	Same

Date: ______________________ Signature: ______________________
Date: ______________________ Approved: ______________________

lay out a project plan. Each phase was subdivided into tasks, with each task being identified by a code number. The methodology defined productivity as rate at which tasks were completed. The time required for each task was estimated and an expected resource requirement was recorded. Time was charged to tasks so that the difference between the expected resource requirement and what had actually been charged became a rough measure of the task's status.

The project schedule was then computed by summing up all the task estimates and dividing that sum by the amount of time the developers were expected to spend working on the tasks. So, a 100-hour task could be assigned to an individual dedicated to the task who would require approximately 2.5 weeks to complete it (assuming a 40-hour workweek). The same task, assigned to a three-person team, each person working half-time on the task, would charge 100 hours to the task after a week and three or four days (100/20 hr × 3 people).

During the first year of use, the Director noted a significant number of hours charged to tasks that were already recorded as complete. When he asked about it, the developers told him that they were just trying to keep the completed work up-to-date. For example, during a coding task, a developer might discover some ambiguity in the requirements statement. The time required to talk with the original analyst or to call the user had to be charged somewhere. The Director thought it was strange to be working on a task that the development plan listed as complete. In addition, in most cases, the estimated resource requirement had already been exhausted, so charging more time to the task would make it look like the task took longer than it did. So the Director set a policy of not charging time to closed tasks and notified the development staff by e-mail (see Exhibit 16).

The DSM-SISS project used the DSM-DM methodology to generate a project plan and lay out a schedule (see Exhibit 17).

After the decision was made to use the Midwest software rather than continue with in-house development, the managers in Applications Development conducted a post-mortem on the project. They knew that the developers were upset and viewed the entire effort as unfair.

Exhibit 16 Time Reports E-Mail

To:	Development Team
From:	Director of Applications Development
Subject:	Time Reports

Just a reminder. Do not charge hours to tasks after the phase review has been completed. Once a task is done and the phase review has been approved, hours need to be charged to the current task codes.

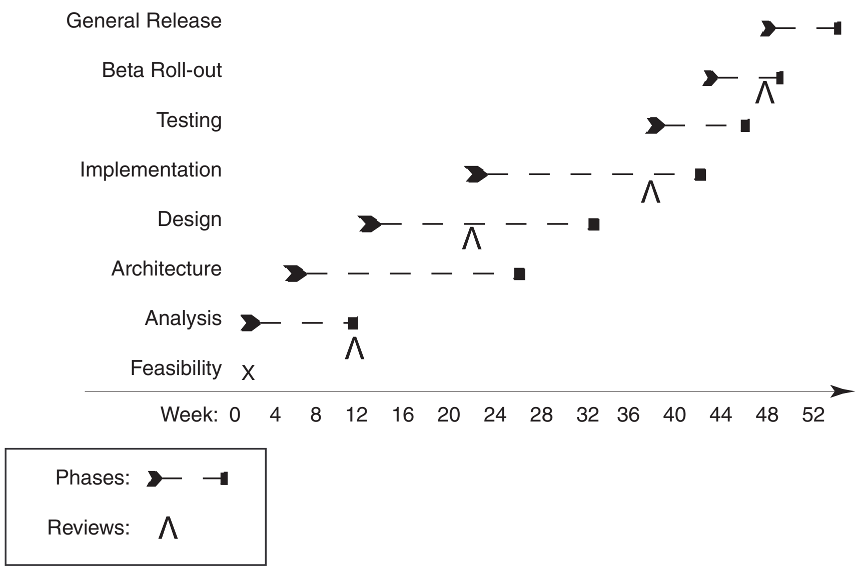


Exhibit 17 Project Overview

During one of the review meetings with the development staff, one programmer said, “We all feel let-down. I went back over my time sheets and the project status reports. We were doing all the right things. Our tasks were coming in almost exactly on budget. We were completing tasks almost exactly on time. So, why did we do all this work only to have the rug pulled out from under us?”

Later, the Director was talking with one of his senior managers: “The programmer is right. How is it that we could look so good on paper and not know that things were so out of control?”

“Well,” replied the manager, “I think we all know in our heart of hearts that the information on the timesheets doesn’t reflect reality. It is as if we ask the developers to tell us what we want to hear. The fact that 100 hours was charged to a 100-hour task does not mean the work is done. Having a review where everyone signs the requirements approval form does not mean that we know what the user really wants nor what they should have.”

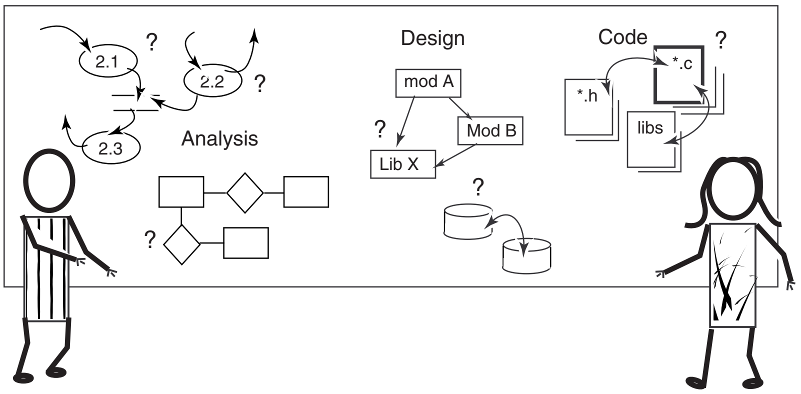


Exhibit 18 Methodology Representation

The Director and the manager went into the conference room. The Director drew some symbols representing the DSM development methodology on the whiteboard (see Exhibit 18). The manager pointed to the "code" phase and said:

> "I think a big problem is that, if we are hit with a change to system (code or design or requirements) at this phase, we don't have a plan for rippling back through the phases to bring everything up-to-date. In fact, we don't even know what might be affected by a given change. So the developers either go hunting through all the past work looking for what needs to be updated, or (more likely) they just change the code and forget about the other stuff. In fact, that policy about time reports tends to encourage just working on the current task because they can't charge time to work we thought was done."

The Director remembered some of the conversations he had had at Midwest. One of the Midwest developers told him that they did not fill out time reports. He thought that was strange, but did not think much of it at the time. He also remembered sitting in on a group session the Midwest developers had in their war room. The Director erased the whiteboard and drew a mock-up of the war room (see Exhibit 19). They had diagrams and printouts hanging all over the walls. Each section of the wall had material from different documents.

The Director continued. "They were talking about a change request. I don't remember the specifics but I remember thinking they were talking about a significant change. One would point to a database schema and list the changes that might be necessary. Another would interrupt pointing to a piece of code and marking it as "needing review." A third would draw some changes on an interface diagram, saying that if the table had to change, this interface should be altered. After a short time they had outlined changes to every document affected by the change. They agreed to get back together the next morning. I assume they left to make the changes. I thought it was strange."

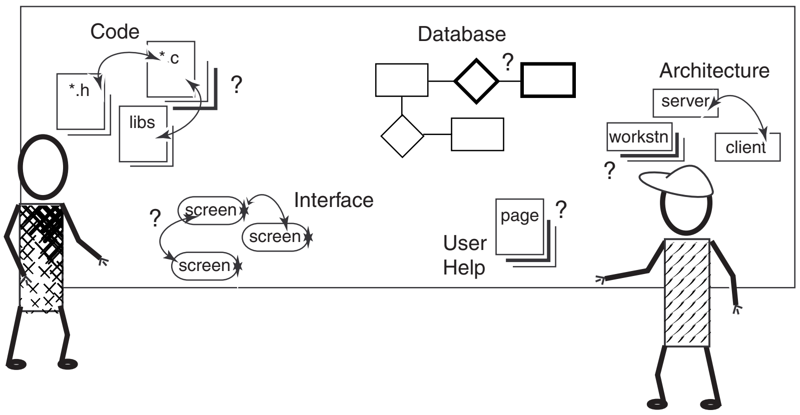


Exhibit 19 Midwest War Room

The Director and manager agreed that some changes were necessary before starting the next big project. "They also said they had no projects," the Director told the manager as they left the conference room. "I'm not sure what they meant by that."

Chapter 3

Planning Progress…

Or What You Don't Know Can Hurt You

Planning the development effort involves visualizing the information you expect to have once all work is completed. The procedural steps and tasks are secondary to shaping and communicating the objective in terms of information that must be created and recorded.

The secret of all victory lies in the organization of the nonobvious.

—Oswald Spengler

Information Structure

At any given time, software development managers are required to define objectives and lay out a direction for the development effort. Teams of developers need to know which elements to build. Senior managers need to know the status of products. Users need to know what new features will be implemented next month. Issues related to strategic planning are discussed in Chapter 6. Here, we need to address tactical planning.

Day-to-day, the manager helps to answer the question: Where do we go today? Month-to-month, the manager helps to answer the question: Where should we go next month? Year-to-year, the manager helps to answer the question: Where will we be next year, next decade?

On the tactical end of the planning spectrum, I am more interested in actions, verbs, and tasks. When I move toward the strategic end of the spectrum, my vocabulary shifts to visions, descriptions, and adjectives.

In my role as a development manager, the focus is on the tactical end of the spectrum. Day-to-day planning assumes you have a policy that says: "In

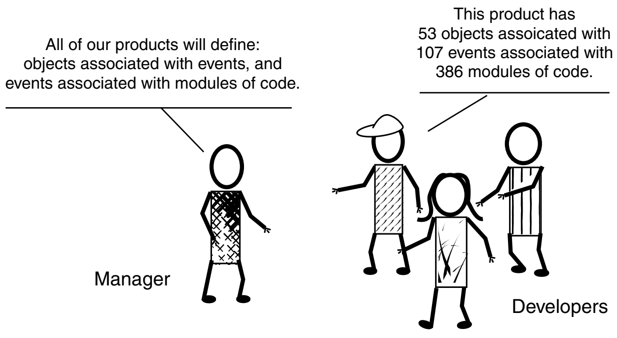

All of our products will define: objects associated with events, and events associated with modules of code.
This product has 53 objects assoicated with 107 events associated with 386 modules of code.
Manager
Developers

Exhibit 1 Planning Dialogue

general, we build certain types of elements." Based on that policy, you and your developers decide what work must be performed to create the expected elements (see Exhibit 1).

> **Planning at a tactical level consists of defining the types of elements developers are expected to build and directing developers to build the elements needed for products under development.**

Tactical planning is an attempt to answers the questions:

- What elements and associations are necessary in order for us to record all necessary information about our products?
- What elements do we need to create for a particular product?

Most development organizations have already defined a standard plan of attack for all new development efforts. For example, either by policy or by convention, there may be an understanding that all products will be defined with an object diagram for recording data object, an event model for each user transaction, and "C" code for each function. The "C" code, event model, and object diagram are examples of Element Definition depicted in Exhibit 2. Defining and establishing this policy is the first part of the planning process. I find it useful to think of this process as defining the structure of the repository (at least the part of the repository where elements will be recorded).

Once the structure is established, you decide what data objects are needed and what functions of "C" code must be written for a given product. These are the tasks and strands discussed in Chapter 2.

Real Building Blocks

The first two chapters used illustrations of element types such as *design*, *requirement*, *test suite*, and *Java class*. While these are common element types, they may not be the elements in which your organization is interested.

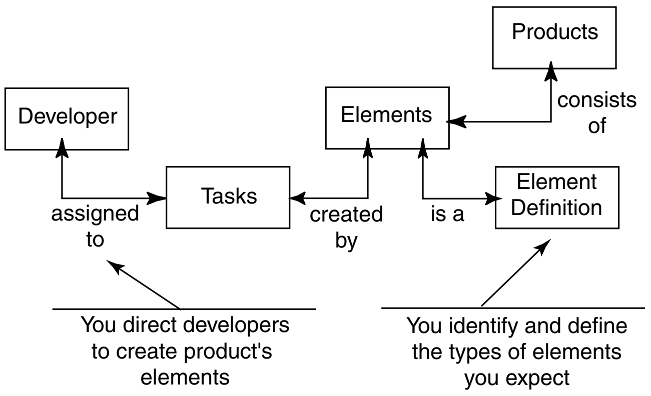

Products
Developer
Elements
consists of
Tasks
Element Definition
assigned to
created by
is a
You direct developers to create product's elements
You identify and define the types of elements you expect

Exhibit 2 Development Repository

Exhibit 3 Examples of Element Definitions

Element Name	*Purpose*	*Weight Unit*
Requirement	Description of an application's properties and behavior expected by a user	Paragraph count
Object Design	Model of a class highlighting public methods and interfaces	Method count Object weight
Entity Design	Definition of a collection of data	Attribute count
Relational Table	Physical structure in a relational database	Column count
Test Case	Definition of initial condition, input, and expected result used to verify an application	Edge count
Test Suite	Collection of Test Cases run in sequence	Scenario count
Java Class	Unit of software compiled with Java used for our intranet applications	Method count Variable count
C++ Class	Unit of software compiled with C++ Compiler used in our Windows/ commercial applications	Member Function count Data Member count
Applet	An executable Java program	Size, in kB
ActiveX Control	Object automating common tasks	Property count

Managers and developers of an organization create element definitions reflecting the information they find necessary to fully describe and understand your products. Examples of element definitions are shown in Exhibit 3. At a minimum, an element definition consists of its name, a description of its purpose, and at least one unit of measure to assess the size of an element of this type.

Your organization will have dozens of elements that have been proven useful in your development work. Your list will be different from that of other organizations. State models will be of great value to teams building process control applications and of little use to teams building data-dominant database query applications. Information system developers usually find entity-relationship models extremely useful, while real-time system developers may have little use for them.

Your list of element definitions will be dynamic as well. A major part of the planning function is the maintenance of the element definitions. As new technologies show their worth, we define new ways to describe them and the applications to which they contribute. Assembler macros might have been on the list of elements a manager expected her developers to build. These might have been replaced by COBOL paragraphs and "C" functions.

> **The set of element definitions describes the building blocks of a generic product.**

It is a way of defining the expectation that the knowledge of any given product will be recorded using these forms. A product is not complete until all necessary elements have been built and verified.

Associations between elements allow you to identify missing information. The fact that a C++ class has been written implies that a design exists. If no design exists, additional work must be performed to find or create it. A requirement typically suggests the need for a data object. If a requirement has been written, the work needed to build one or more data object definitions is accrued. These associations must be defined as part of the planning process.

> **An association definition consists of references to the associated elements, a description of the meaning of each association (from both element's perspectives), and an average frequency.**

The frequency represents the approximate number of associations of this type in which an element will participate and is used to calculate estimates. You will see how the frequency is used later in this chapter. Examples of association definitions are shown in Exhibit 4.

It is easiest to think about association definitions as sentences describing the "rules of the game." The rows in Exhibit 4 can be transcribed into prose as follows:

> On average, a *Requirement* suggests data defined as two Object Designs, and an *Object Design* defines the data suggested in five Requirements. An *Object Design* is stored as two Entity Designs, and an *Entity Design* describes the persistent structure for one Object Design. An *Entity Design* is implemented as two Relational Tables, and a *Relational Table* implements one Entity Design. A *Java Class* implements one Object Design, and an *Object Design* is implemented as one Java Class. A *Java Class* is exercised by ten Test Cases, and a *Test Case* exercises one Java Class. An *Applet* uses one Java Class,

Exhibit 4 Examples of Association Definitions

Element	*Meaning*	*Avg. Freq.*	*Element*	*Meaning*	*Avg. Freq.*
Requirement	suggests data defined as	2	Object Design	defines data suggested in	5
Object Design	is stored as	2	Entity Design	describes persistent structure for	1
Entity Design	is implemented as	2	Relational Table	implements	1
Java Class	implements	1	Object Design	is implemented as	1
Java Class	is exercised by	10	Test Case	exercises	1
Applet	uses	1	Java Class	used in	2
Test Suite	runs	40	Test Case	run by	1
Test Suite	exercises	1	Applet	is exercises by	1

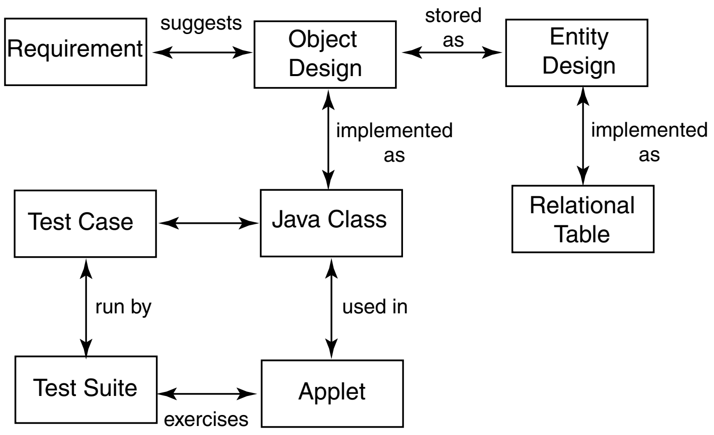


Exhibit 5 Element and Association Model

and a *Java Class* is used in two Applets. A *Test Suite* runs 40 Test Cases, and a *Test Case* is run by one Test Suite. A *Test Suite* exercises one Applet, and an *applet* is exercised by one Test Suite.

Graphically, the model of the element and association definitions is depicted in Exhibit 5, the type of model supplied by CASE vendors to describe the information managed by their Computer Aided Software Engineering tools. It is also the type of model used to describe development methodologies. CASE tools are partial implementations of the development repository. They allow developers and managers to record development knowledge and assist in assuring consistency. Methodologies are generic project plans defining the steps to be performed in typical development efforts and the output of each step. The output definitions are synonymous with the element definitions described here.

If you are using a CASE tool or a development methodology (and if they are useful to you), you have definitions of most of the elements you need for planning purposes. The reason I include the qualification is that different organizations find value in different sets for elements and association. Their CASE tool or methodology may not support all of these elements and associations. Similarly, the CASE tool or methodology may call for elements that are of little or no use and should not be included in the planning process.

For example, one organization used a data modeling tool that required the data analysis to include a physical definition of data attributes (size, data type, and language-dependent name) when an entity was declared. An error occurred if at least an identifier and one non-key attribute did not accompany the definition. This made it difficult to define an entity (a logical representation) and a relational table (an implementation) as two separate but associated elements. The analysts used a drawing package to maintain the logical view. If you were using the data modeling tool as a source of element definitions, you would have to look beyond the tool to the way the tool was used and to the additional information the analysts determined was valuable.

While some tools are missing elements that developers find valuable, the reverse is also true. Many tools and methodologies require the creation of elements that are less than useless to the developers. A CASE tool I once used had the bizarre feature of requiring that the developer create a data definition each time a new data store was declared and again each time a process accessed the data store. There were hundreds of processes accessing the same data store; 95 percent of them were accessing a single occurrence of the data store. This meant there were potentially hundreds of redundant definitions of the data.

Part of the planning process involves asking which chunks of information are necessary and sufficient for a complete, useful, and maintainable repository of knowledge about your products. Being included as part of a software engineering tool or being recommended by a textbook author does not, in itself, mean the element should be part of your repository. You and your developers have to make realistic decisions about which chunks of information are worth the investment of time, money, and brainpower. Like all assets, their value comes in their use.

Where do you find potential element and association definitions?

- Your current methodology and practice
- Old documentation
- New books on project management and software development
- Old books on project management and software development
- Observation of developers enhancing existing products

I prefer direct observation. There is no practice more useful to observe than the maintenance process. If you came up through the ranks in software development, you probably started in maintenance.

Exhibit 6 Sample Element/Association Definitions

Element	*Meaning*	*Avg. Freq.*	*Element*	*Meaning*	*Avg. Freq.*
Requirement	suggests data defined as	2	Object Design	defines data suggested in	5
Java Class	implements	1	Object Design	is implemented as	1

Starting new programmers on maintenance is a horrible practice! The most destructive beginning for a new developer is being turned loose on an existing product and told to "fix it."

Remember back to that experience and you will remember a high level of uncertainty. You did not know what the original analyst had in mind. You were unsure of the user's expectations. You did not know if the code you were changing was the only code that needed to be changed. You were never sure that you were not introducing additional bugs into the product while trying to correct someone else's errors.

Most developers spend a great deal of time trying to rediscover information that was known at one time but now is lost or inaccessible. By observing the maintenance activity, you can identify elements that will record the information your developers need in order to build and enhance your company's software products. Chapter 7 explores in greater detail the process of building and maintaining a productive development environment.

By creating elements and associated definitions, you are creating a framework of work products. As an example, assume you have defined the elements and associations described in Exhibit 6. One of your developers discovers the need for a new Object Design. An important question is: How much work is implied by the addition of this one new Object Design? (See Exhibit 7.) The answer is that your developers will have to create:

- Five associations to Requirements (probably necessitating changes to the Requirements)
- One new Java Class and its association

Work with your developers to create an initial set of element and association definitions. The frequencies help you anticipate the amount of work accrued by the addition of a single element. Because information is meaningful only in context, the new information created by the addition of a single element will not be useful until it is associated with related elements. If the typical Object Design in your organization is associated with five Requirements, the new Object Design implies that work must be performed to build associations to (and potentially modify) about five Requirements as well as to build the Java Class to implement the design.

The average frequency (sometime referred to as "cardinality") of an association helps predict the number of things that must be done when changes

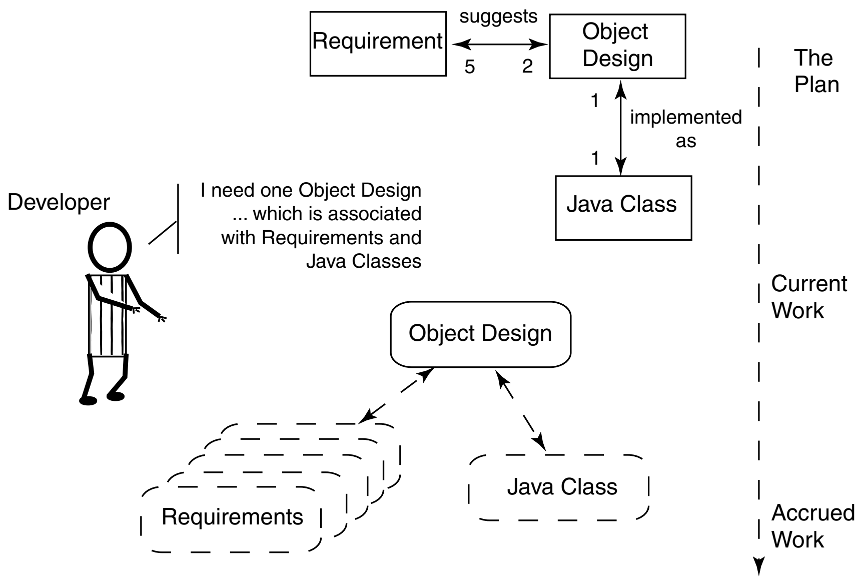


Exhibit 7 Using the Plan to Predict Work

are made to the repository. The size of the elements is needed to be able to estimate the amount of work involved. You might assign tasks to two developers to building a C++ class based on object designs. If one developer's object design is twice as large as the other's, you might expect one to take twice as much time (assuming both developers are equally experienced and talented with C++). Chapter 5 explores the formulae useful in monitoring productivity.

For now, the planning process includes identifying the unit of measure you need to monitor the "size" of the elements. Refer to Exhibit 3 for examples. We are looking for units of measure with a scale that has a positive relationship to the amount of resource (usually time) required to define and verify an element. In the ideal state, an element that has a weight of five will require half the time to define and verify as an element of the same type with a weight of ten.

We are also looking for units of measure that are predictive. If we cannot determine a reasonable estimate of the element's weight until after the element is completely defined and verified, the weight has no use to us (other than historical).

> In the absence of any better definition, they understand an estimate to be the most optimistic imaginable result that is not demonstrably impossible. That kind of estimate is a disaster for planning purposes.
>
> **—Tom DeMarco**

Be mindful of the difference between size and value. Software development managers have had a long-running debate about the role and use of counts like "lines of code." It has the advantage of being easily computed. It has the distinct disadvantage of having no valid relationship between its scale and the value of the product. That is, a module of code that has 500 lines of code is not twice as valuable as one that has 250 lines of code. It might be true that of two modules performing the same function, the one with the fewest number of lines is more valuable.

If you equate the scale of a measure with "good work," you encourage developers to produce elements that rank high on the metric's scale but may add little value to the repository and the end user. We tend to work to maximize the metric against which we believe we are being evaluated.

Reward Complete Thinking

With a set of element and association definitions in place, the next aspect of tactical planning to consider is deciding which elements to work on at any given moment. As the workday progresses, developers face countless alternatives. After completing the definition of a requirement, a developer might proceed to the requirement's design or start working on a related requirement. A developer might discover an ambiguity in a code module after running a test case and might either work to correct the code module or continue building additional test cases.

Perhaps the decision is unimportant because all the work has to be completed eventually. However, different sequences result in different degrees of uncertainty. In addition, while I believe developers should work with a significant degree of autonomy, I do have to define direction and overall patterns to ensure the completion of the most constructive and beneficial work.

Rewarding complete thinking is the most effective way of establishing a constructive direction for the development effort at the tactical level of planning. By this I mean defining the types of strands I expect the developers to be working on. The message to developers is: if there are incomplete elements of the strand, those elements take precedence.

The most useful strand is the one that provides a complete description of a single event and its implementation. From the user's perspective, the software system performs sets of tasks. If the user needs enhancement to a Product Inquiry function, the strand on which a developer will be working might include a Class Design, several Java Classes, the definition of the Product Data, and the Inquiry Requirement (depicted in Exhibit 8). As discussed in Chapter 2, the strand is a useful definition of work. Each strand corresponds to a task assigned to a developer. The amount of work remaining in the task corresponds to the number of unverified elements adjusted by their weight.

There may be occasions when you will need to plan a task to declare and define a large set of a particular element type (see Exhibit 9). For example, if you are planning a significant enhancement to applications supporting the purchasing department, it might be useful to have at least an initial declaration of all the department's anticipated data objects. This can provide a useful overview of the scope of the effort and a good starting point for defining more detailed strands.

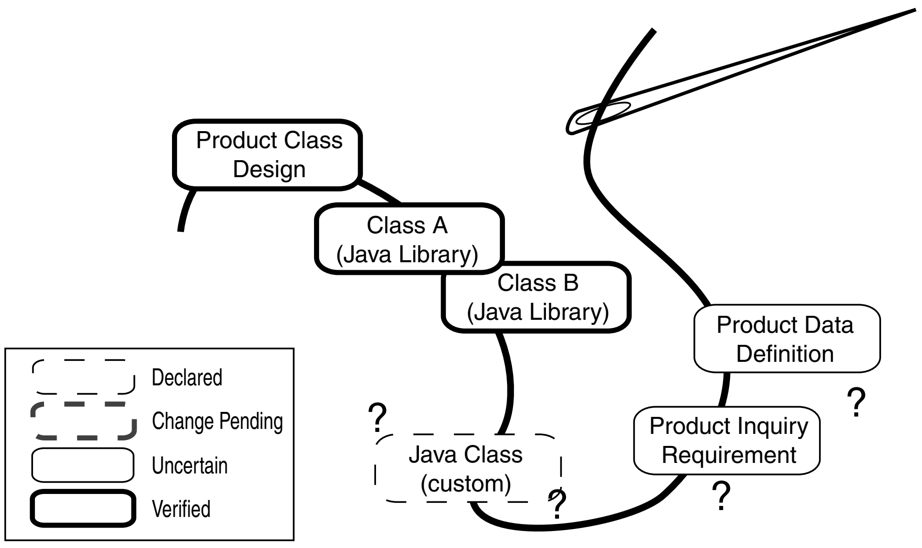


Exhibit 8 Three Down, Three to Go

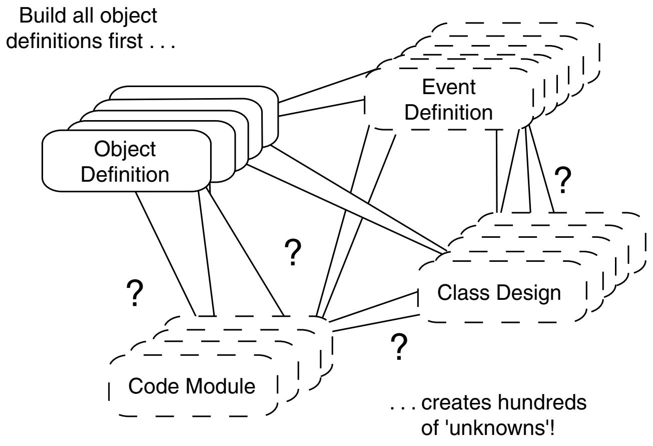

Build all object definitions first . . .
Event Definition
Object Definition
Class Design
Code Module
?
?
?
?
. . . creates hundreds of 'unknowns'!

Exhibit 9 Declaring All Instances of One Element

The danger in declaring and defining elements without defining associated elements is that each declared element implies that work must be done to define and connect all the related elements. Each of these undeclared and unconnected elements is an unknown. The greater the number of unknowns, the greater the risk. We are accruing more and more work without evidence that it will fit together in the end. We are creating expectations that will be more and more difficult to manage.

Real Uncertainty

Developers, like most humans, attempt to make sense out of incomplete information (either by finding the missing pieces or by rationalizing the inconsistencies). Rationalization is the easier option, but actually building the missing pieces is certainly more valuable. We tend to underestimate the complexity of future work. We tend to be optimistic about how much we know and how easy it will be to fill in the blanks.

> Developer 1: "I have been getting good information from the user. The Object Definitions are coming along great!"
>
> Developer 2: "So, when should we start building the Event Definitions and Code Modules?"
>
> Developer 1: "Those will be straightforward. I'm not worried about them. Once all the Object Definitions are completed, we can wipe those out in no time."

Conversations like this are common and often an indication that the development effort is out of control. A month later, you hear the same two developers.

> Developer 1: "These Event Definitions are more complex than I expected."
>
> Developer 2: "I think there are some problems in the Object Definitions. If we had known about some of these business rules before, we could have defined some of the Objects differently and saved ourselves some headaches."
>
> Developer 1: "Well, the Object Definitions are done, so let's not open that can of worms."

The result of this line of thinking is that the Object Definitions are incomplete, inaccurate, or inconsistent with the Event Definitions. The inconsistency between the Object Definition and the Event Definition implies the need for work to correct the error. The repository no longer reflects what we know about the product. Future work will be more error-prone due to the misinformation.

> Success is more a function of consistent common sense than it is of genius.
>
> **—An Wang**

The developers should be allowed and encouraged to think of their work as the process of moving the repository from one steady state to the next, rather than simply adding more and more information. Thus, any planning should focus on the desired state rather than tasks.

Watch the Result, Not the Process

Chapter 2 defined a unit of work as creating chunks of information; specifically, creating or enhancing a strand of elements. A strand is the purpose of a task. A developer (or group of developers) is assigned a task of building or

Exhibit 10 Development Plan

Task ID	*Focus Element*	*Element Count*	*Description*	*Priority/ Sequence*	*Assigned To:*
A	Customer Object Design	20	Add customer order status	Done	Blue Team
B	Customer Object Design	12	Add customer credit query	Done	Clayton
C	Product Object Design	23	Enhanced product pricing formula	1	Blue Team
D	Product Object Design	8	Replace inventory valuation method	2	Clayton
E	Product Object Design	17	Add product inquiry	3	?

enhancing a strand of elements. As a manager, I have little concern for how the information is created. I coordinate the development effort by defining the tasks and monitoring the repository for the status of the task.

So, what does a development plan in a dynamic environment look like? A plan consists of the focus element of the strand, along with a brief description of the functionality or value to be added to the application. For each task in the plan, you estimate the number of elements that may be affected by the developers while working on the task. Each identified task is prioritized or sequenced and assigned to developers working alone or in teams. Exhibit 10 presents an example. Tasks C and D have been assigned to developers. The Blue Team is working on enhancing product pricing formulae and Clayton is working on replacing the inventory valuation method. Task E has not been assigned. Two customer-related tasks have already been completed. Task A was performed by the Blue Team. They modified or created 20 elements while adding functionality to report the status of customer orders. While Clayton was adding a query for customer credit, he modified or created 12 elements.

Typically, the plan does not include guesses as to when a task will begin or how long it will take. A task will begin as soon as developers have completed tasks of higher priority. The work will take approximately the same length of time as other similar tasks. You rely upon the information in the repository to compute estimates of dates and resource requirements.

For example, Exhibit 11 contains information about the dates and person-days for the two completed tasks (see also Exhibit 12). The Blue Team started Task A on the first day of the month. They completed it on the third day and charged ten person-days to the task. Clayton started work on Task B on the first day of the month and completed it on the fourth day, requiring four person-days. In terms of elapsed time, the Blue Team was working at a rate of between six and seven elements per day (6.67 to be more precise). Clayton completed work on 12 elements in four days (three elements per day).

If today is the fifth working day of the month, the Blue Team started working on Task C yesterday and Clayton begins work on Task D today. Based on the information in Exhibit 11, both the tasks will probably be

Exhibit 11 Work Record

Task ID	*Developer*	*Started*	*Completed*	*Person-days*
A	Blue Team	1	3	10
B	Clayton	1	4	4

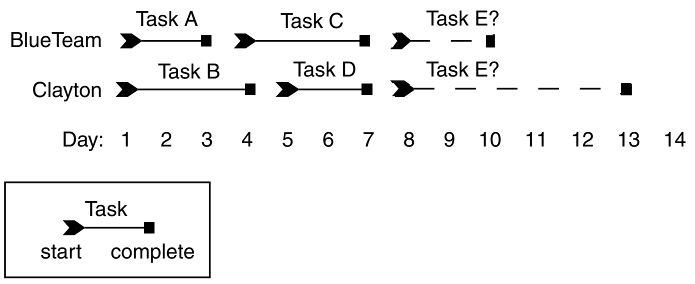


Exhibit 12 Development Schedule

completed on the seventh working day. You know this because Task C is projected as effecting 23 elements. The Blue Team has been working at a rate of approximately six to seven elements per day (elapsed time). Task C should take them about 3.5 days. They started yesterday so they will be done mid-day on the seventh working day. Clayton's task will require creating or modifying approximately eight elements. At three elements per day, Clayton will be done in 2.6 days — call it three days.

The unanswered questions concerns Task E. If the time projections for Tasks C and D prove to be reasonably accurate, Clayton should be able to complete Task E before the 13th or 14th workday of the month (17 elements divided by three elements per day suggests about five days). Assigning task E to the Blue Team will get the work done by the tenth day (17 elements divided by 6.7 elements per day equals 2.5 days).

The decision to assign Clayton or the Blue Team to Task E is not only a function of the projected elapsed time. Other factors include skill and expertise of the developers (more on this in the next chapter) and other tasks pending on the development plan (discussed in Chapters 9 and 10).

Applying Dynamic Management

Work with your developers to expand the initial list of elements you started in Chapter 1. Add a column to the element worksheet for a "Weight." Discuss each element's purpose and list measures that might be used to determine size.

Talk about the associations among elements that the developers find important. The set of elements and associations are the framework on which your development plan is based (see Exhibit 13). The strands that are part of

Exhibit 13 Blank Element List

Element Name	*Description, Location, Associations*	*Weight Unit*

the task definitions are determined by the elements and associations you list. The list does not have to be absolute. You and your developers will refine the definition as your environment develops and your applications evolve.

Review the work you and your team have completed in the past month. Describe this recently completed work in a form similar to Exhibits 10 and 11. This helps describe the tasks as strands of elements with a focus. Estimate the number of elements you worked on (i.e., created or modified) during the month and calculate the rate (elements per workday).

Productivity = Number of created or modified elements
per number of person-days

This number may not be precise, but it is consistent, useful, and easy to generate without the biases inherent in other forms of projections.

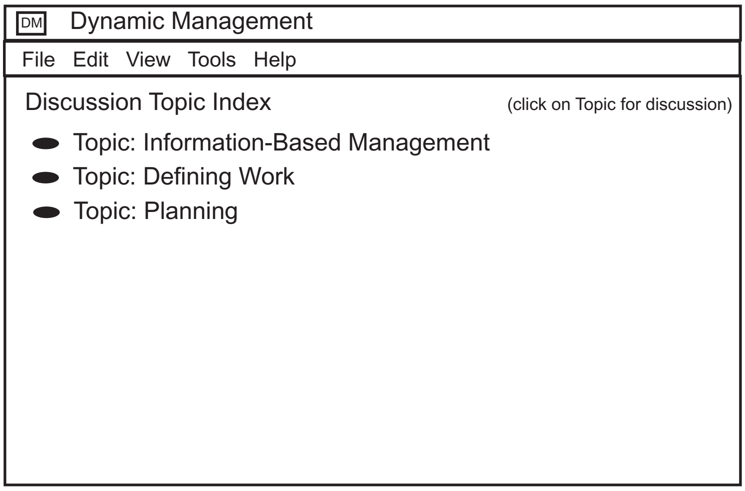
Add a topic to your online discussion group (Exhibit 14) and attach a copy of the worksheet. Ask your peers for comments and suggestions (Exhibit 15).

Chapter 3 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

In the early years, as DSM development efforts grew in size and complexity, managers and developers alike recognized the need for more structure in the way software was built. Industry trade journals were filled with articles debating various methodologies and their characteristics: waterfall, spiral, model-driven, and process-oriented methods all focused on "process." Consulting firms preached analogies to other engineering and manufacturing practices. The managers and developers at DSM adopted the analogy of building a house with the need for careful plans and blueprints. Construction began only after careful planning; a firm foundation is laid before the walls and roof are added. Enhancements identified after construction begins are postponed and handled by workers under a different contract.

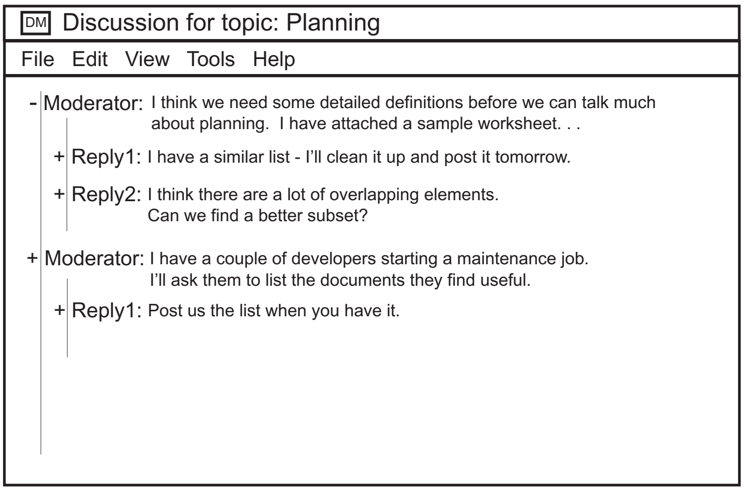
DSM bought into the current thinking of the day and wanted a standard methodology for all its new development projects. Maintenance was to be treated as a mini new-development project. DSM contracted with a consulting firm to help define the methodology to be used in all its development. The consultants worked with the managers and senior analysts to try to customize the consulting firm's template methodology as best they could. At the end of



Discussion Topic Index (click on Topic for discussion)

- Topic: Information-Based Management
- Topic: Defining Work
- Topic: Planning

Exhibit 14 Online Discussion Index



- Moderator: I think we need some detailed definitions before we can talk much about planning. I have attached a sample worksheet. . .
  + Reply1: I have a similar list - I'll clean it up and post it tomorrow.
  + Reply2: I think there are a lot of overlapping elements. Can we find a better subset?

+ Moderator: I have a couple of developers starting a maintenance job. I'll ask them to list the documents they find useful.
  + Reply1: Post us the list when you have it.

Exhibit 15 Beginning Dialogue

a difficult six months, the methodology was delivered in 17 beautiful binders. DSM paid the consultants a small phenomenal fee and set about the task of using the new procedures they had defined for themselves.

All was OK until the disaster of the DSM-SISS project. After that, the development department entered a period of adjustment. Most of the DSM-DM methodology fell into disuse. Project managers were allowed to pick and choose those tasks that seemed most appropriate. As time went on, fewer and fewer of the tasks were considered "appropriate."

The support and enhancement of the DSM-SISS remained with the Midwest group. Development on the West Coast continued with other independent

Exhibit 16 E-Mail to Boss and Reply

E-mail

To: Midwest Development Manager

From: Concerned Developer

Subject: Upcoming DSM Development Conference

I am really concerned about the upcoming conference. I see no reason for our parent company to impose its formal and stifling methods on us. After all, it was our product that got them out of a bind several years ago. How should we respond to this?

Reply

To: Concerned Developer

From: Midwest Development Manager

Subject: DSM Development Conference

I think this is a good chance to build some bridges and expand our own influence in the larger organization. We should go with an open mind. We will take what works and change what doesn't. My advice to all of us is to take every opportunity to show the good work we are doing. Make as many contacts as you can and speak out in favor of the techniques we use in building successful products. I think the people at DSM are open to innovation.

products but more and more time was spent in maintaining and upgrading existing systems.

About five years ago, it became increasingly clear that the products being developed by DSM and Midwest had to be updated and integrated. The Director, now the Chief Information Officer, had wanted to bring the best of both development groups together. He felt it was time, at long last, to do something significant and positive. He initiated his campaign by convening a development conference. A week was set aside for all developers to gather in San Francisco to come up with a plan for developing closer working relationships. Sessions were planned to provide the Midwest group with detailed orientation to the DSM-DM methodology, the current set of standards used on the West Coast, and the development environments currently in use.

When the developers at Midwest received the announcement of a "conference," they immediately concluded that DSM was finally going to indoctrinate them into their stiff and formal methodology.

One of the Midwest developers e-mailed her boss, and the manager of the Midwest development group replied, as shown in Exhibit 16.

So, the conference took place. Much of it was boring. The two sets of developers began taking longer and longer lunch breaks to exchange stories about past projects. They even convened impromptu workshops to learn each other's development secrets. Both the DSM CIO and the Midwest manager agreed that the informal exchanges were probably more valuable than the

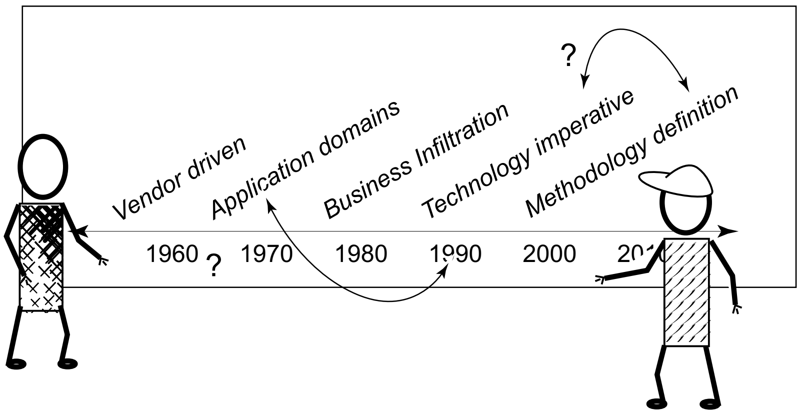


Exhibit 17 Development Eras

planned sessions so they cut short the formal training in favor of the more spontaneous sessions.

During one of the breaks (i.e., spontaneous sessions), a large group was gathered in the conference room discussing where the application development industry had come from and where it was going. One of the DMS developers went to the whiteboard and drew a timeline, saying he was happy to see the passing of the era of vendor-supplied, bundled applications. Someone pointed out that maybe things have not changed that much. The big ERP implementation projects are painfully reminiscent of those days — maybe the era never ended (see Exhibit 17). Someone else grabbed a marker and wrote, "Application domains," saying she was glad we are developed beyond data silos. A voice from the back said, half jokingly, "Hey, those were some of my best work."

One of the project managers wrote "Business infiltration" to represent the shift from discrete systems to systems that better integrated into the daily business operations. That prompted one analyst to write "Technology imperative." "It seemed that along with the business integration came the sense that technology was the end-all solution to everyone's problems. Everyone has to have the latest version and the most advanced hardware. I don't think the latest version necessarily adds value." There was overlapping debate over which era came first and if any really ended. At one point there were five people at the board, each armed with a marker.

"What about development methodologies?" came a voice from one of the junior developers. No one had wanted to touch the subject of methodology, but apparently the kid did not know any better. The CIO walked up to the board and wrote "Methodology definition." This legitimized the subject, so people began to talk.

The group began exchanging ideas about how development projects should be partitioned into manageable steps. Some were dutifully defending the phased approach while others began to assert that there were major problems with the construction or the factory analogy. After a couple of minutes, the white board had been erased and a new image was being drawn.

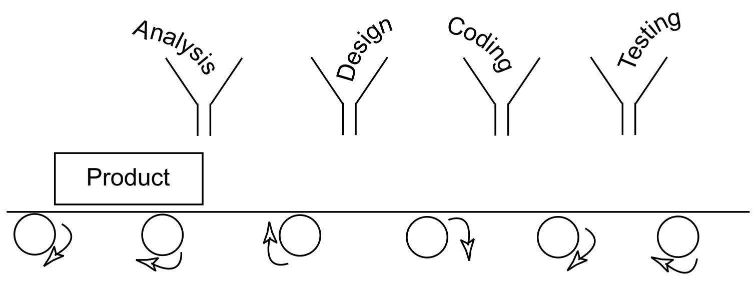


Exhibit 18 Methodology as an Assembly Line

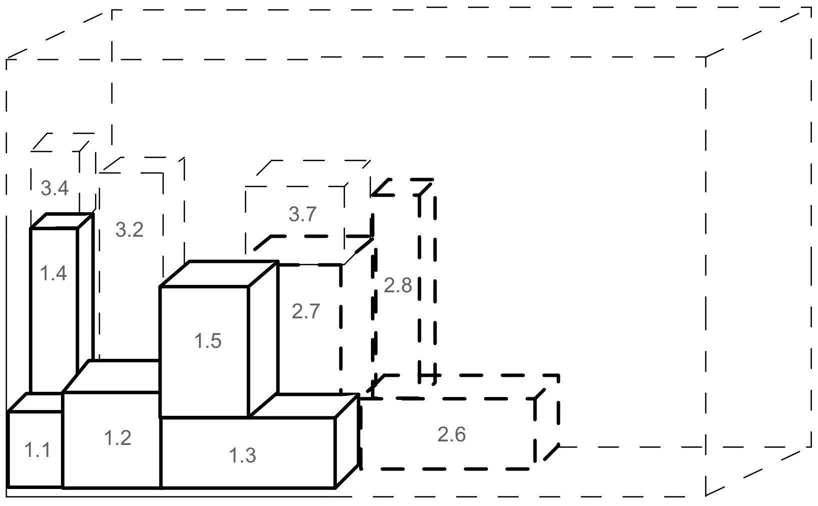


Exhibit 19 Methodology as a Continuous Build

It was clear that the general consensus was that project methodologies seemed to impose the feeling of an assembly line (see Exhibit 18) and that image did not feel good to most in the room. The drawing on the board took on the mantle of satire. The conversation began to slow.

The Midwest manager stepped forward. "What if we turn things around?" he said, erasing the board again. "The model we have been discussing seems to divide the work of a project by the type of work first, then the individual features, functions, or objects. Maybe we can divide up the development effort by the object, function, or feature first and then subdivide that by the type of work — requirements, design, coding, etc."

The Midwest manager began to draw a huge cube that filled the whiteboard (see Exhibit 19). All the lines were dashed except the lower left-hand corner. "Let's say the box here is the product. It is mostly undefined. We can imagine the whole thing but only in our imagination." "Down here," referring to the solid lines, "we have what we know for sure. We know enough to define a first version — say, chunks 1 through 5. We think we can add chunks 6 through 8 in the second version. We might even be able to foresee improved versions of chunks 2, 4, and 7 — but we won't get to them until a third version. Of course, these future versions will actually be determined by the feedback we get from the users when they start using the first version."

The CIO recognized the drawing from the Midwest conference room conversation several years before. Obviously, the manager had spent a lot of time thinking about this. "So, what are these blocks?" asked the CIO. "They are features or functions or scenarios or requests," replied the Midwest manager. "They are cohesive chunks of the product that the user has requested." "But what work products do each of the blocks represent?" demanded the CIO. The Midwest manager responded "They represent our understanding of the user's requirements (written in the form of use cases and object definitions), our declaration of how the requirements will be implemented (in the form of database tables and code/class structure), the code satisfying the requirements, the test scenarios used to exercise the product, the user training updates, and help systems reflecting the new enhancements. Each version adds value/functionality and keeps all the information current. This way, the objective is not to move a complete system along the assembly line as fast as possible. The objective is to keep adding value — piece by piece — forever."

There was silence in the room. The CIO was recalling the conversation he had had a few years earlier with the Midwest development manager. After a moment, the CIO said, "Well, OK then. Let's work on it."

Chapter 4

Managing Developers…

Or Dance with the One Who Brought You

Chapter 2 outlined a definition of work as the process of creating complete, consistent information about the product. Chapter 3 defined a way those units of work are organized as part of a tactical plan. This chapter (Chapter 4) addresses the work of managing and coordinating talent.

> What is the real reason why we want to be big, to be creative geniuses? For posterity? No. To be pointed out when we stroll in crowded places? No. To carry on with our daily toil under the conviction that whatever we do is worth the trouble, is something unique — for the day, not for eternity.
>
> **—Cesare Pavese**

No Management? No Documentation?

When I have asked software developers to describe what management can do to help improve the development effort, I am usually told to "have management leave us alone and the work will get done." There is widespread belief that management is not real work and that real work is hindered by attempts to manage it.

Most of my manager colleagues do not appreciate the advice. Managers know that their role is important and that the development process would work more smoothly if work could be planned and organized more effectively. My observations are that most managers bring value to the equation. This value is often not recognized by developers.

However, the comment is valuable insight into the developer's perception. Developers often view management activities as impediments to real progress. Most developers with whom I have worked want very much to do good work, advance their knowledge and skills, and be recognized as contributing to worthy objectives. To the extent that management structures detract from the developer's goals, developers and managers will continue to have a contentious relationship. To the extent that management structures support these goals, the relationship between developers and managers will be constructive.

I have read comments in several extreme programming online discussion areas expressing a distrust or disdain for traditional management. I have also noticed many comments casting aspersions on "documentation" — as if it were an institution. I recall one comment asserting that documentation is a burden, but recognizing that some form of memory is required. The comment went on to assert that the team's collective experience serves this purpose.

The coordination of talented developers has been likened to herding cats. My goal as manager is to create a framework in which there is good fit between the information I need to monitor, track, and measure the development effort and the information used and created by the development teams. That fit is assured by building and maintaining an accurate model of elements and associations, then monitoring the contents of the project repository. This way, both managers and developers are focused on the same information. Progress toward the goal (i.e., the ongoing delivery of high-quality, responsive, and effective software to the user) is accurately measured by actual work products of the development teams.

Managing developers starts with recognizing the common view that managers get in the way and documenting is simply busy-work. To the extent this view reflects reality, an organization is operating at a severe disadvantage.

The collective goal of all those involved in a development effort must be to build and maintain a dynamic environment in which developers and managers can be creative — on a daily basis — where managers and developers can provide value to their organization — on a dailby basis — and where managers and developers can grow in their profession and in their personal lives.

We will make no progress toward this goal if developers insist that their personal memory and ability to communicate constitute a sufficient development repository. We will not advance toward this goal if managers try to impose burdensome reporting requirements separate from the knowledge necessary for effective software development.

Diverse Skill Set

The definition of work presented in Chapter 2 requires a diverse set of skills. A strand may include requirements information, data object design information, coding information, testing information, and educational material information. All this information must be created and reviewed to ensure that it is consistent with other elements of information in the repository. No single developer has

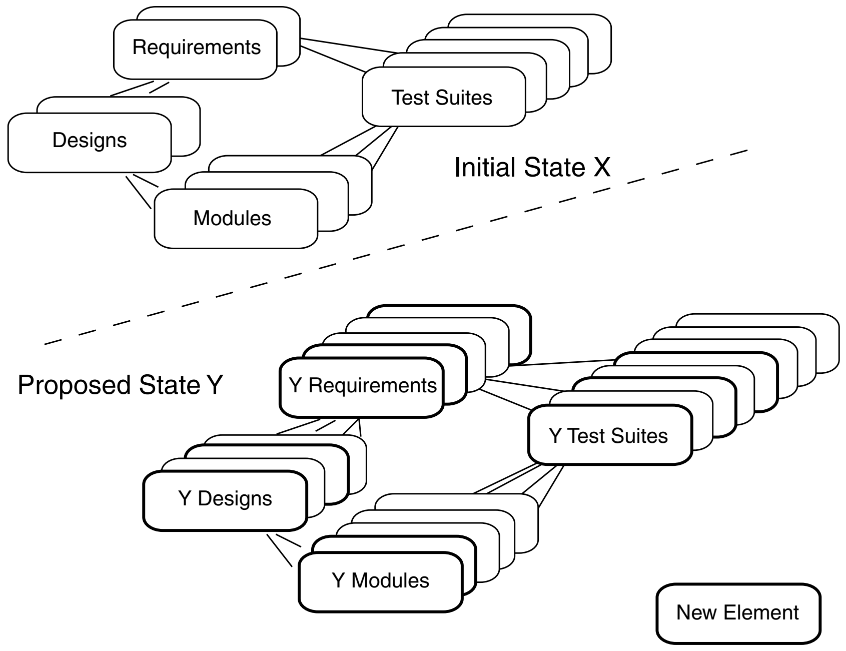


Exhibit 1 Initial State to New Steady State

the expertise or interest in all these forms of information — nor will one developer have the ability to switch from role to role in the short time frame in which the strand is being developed.

Software development proceeds most effectively when done by small teams of specialists, each with a heartfelt appreciation for the skills and contributions of others on the team.

Visualize an application with an internally consistent and verified set of elements. This version of the application is in production and being used by many people in the organization. A new/expanded functionality set has been identified and the developers have initially identified a strand of ten elements, including:

- Three new requirements elements
- Two new design elements
- Two modules
- Three additional test suites

The goal of the development effort is to build the new elements and ensure consistency with the elements already in the repository. The team will move the knowledge in the repository from one steady state (State X) to another steady state (State Y) (see Exhibit 1).

To accomplish this work, you need developers who can build requirements, designs, related modules of code, and test suites that exercise the other elements in an effort to find defects. I am not concerned about which element is created first. Indeed, the elements must be created in tandem. An error

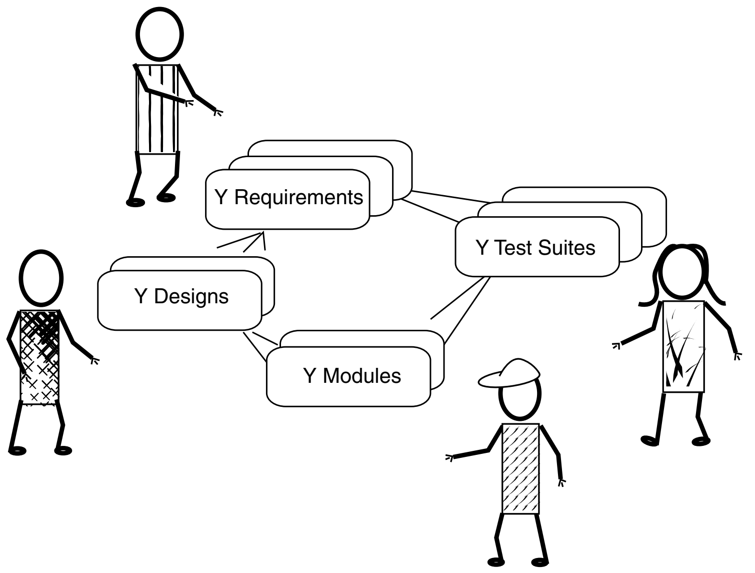


Exhibit 2 Team Assigned to Task (State Y)

Exhibit 3 Expanded Element/Association Definitions

Element	*Meaning*	*Avg. Freq.*	*Element*	*Meaning*	*Avg. Freq.*
Requirement	suggests data defined as	2	Object Design	defines data suggested in	5
Java Class	implements	1	Object Design	is implemented as	1
Test Suite	exercises	1	Java Class	is exercises by	2
Requirement	describes	1	Test Suite	is based on	1

detected by a test suite may imply an error in the code. It may also suggest an error in the requirements. The test suite itself may be in error. Rather than dividing the work across time and across different skills, the work must be done by a close-knit team watching and participating in the entire process of moving the repository from State X to State Y (see Exhibit 2).

Teams of two to seven developers work well. The teams should include individuals with the skills necessary to build strands defined in the planning process (refer to Chapter 3). Visualize a team of four developers: a business analyst, a software designer, a programmer, and a tester. For this illustration I will augment Exhibit 6 in Chapter 3 where I presented an example of an element/association definition. Exhibit 3 adds definitions for the test suite illustrated above.

Given these definitions, when we identify the need for two new requirements, our repository speaks to us by projecting that each requirement is typically associated with two object designs (the object design may be associated with other requirements already implemented). Each object design is

Exhibit 4 Team Responsibilities

Team Member	*Responsible For:*	*Working With:*
Business analyst	Two Requirements	The software designer to define the objects The tester to build the test suites
Software designer	Four Object Designs	The business analyst for requirements The programmer to build the classes
Programmer	Four Java Classes	The software designer for specifications The tester to build test cases
Tester	Four Test Suites	The programmer for interface specifications The business analyst for scenarios

usually associated with one Java class. We expect each Java class to be exercised by two test suites. And it is typical for each test suite to be based on one requirement. At the beginning of the task, the developers can expect to be working on a total of 14 elements (two requirements, four object designs, four Java classes, and four test suites). During the task, the developers might find the need for more classes or fewer test suites, but based on the current state of the repository, the expectation is 14 elements. The work of the team should be distributed as shown in Exhibit 4.

The business analyst is responsible for the definition and overall quality of the requirements elements. But part of the characteristics of good software is that requirements are consistent with other related knowledge; specifically, the object designs and test suites. So, the business analyst must work with the person responsible for the object designs and test suites. Similarly, the programmer needs to work with the person who designs the objects and the person building the test suites. The programmer is responsible for the quality of the Java classes and is obliged to work with others on the team to ensure consistent knowledge in the repository.

If the contents of the repository suggest that a strand will have a larger number of one element, the typical team size would change. For example, if the typical object design was associated with three Java classes and each class was exercised by four test suites, the number elements in this example balloons to 66 elements (two requirements, four object designs, 12 Java classes, and 48 test suites). This suggests that an effective team might include two programmers and several testers.

As task sizes get larger, there is a tendency to form larger teams. But teams formed to do creative work become less effective as they get larger. I find that teams of eight or more self-organize into smaller, more manageable partnerships. Rather than trying to impose some organizational framework on larger teams, it is better to keep the tasks smaller and the team membership below seven. For the larger tasks, team members support each other. While the tester may still be responsible for 48 test suites, the business analyst and the programmer can assist by building test suites along with the tester.

Combining Work and Learning

One of the wonderful benefits of team assignments is the opportunity for team members to learn from each other in the context of real and meaningful work.

You surely remember that moment shortly after you started your first development job when you realized that what you were taught in school is not a close match to what you perceive on the job. Things look different when you are in the midst of a development effort, far different from the way they did in the safety of a classroom. You probably also had the same experience I did when my first assignment was to enhance an existing program. I was terrified because the number of things I did not know was overwhelming. Working alone, I studied the program, read the documentation, and worked hard to emulate the style and techniques of the developers who had built the original product.

It was years later that I realized that the style and techniques of the developers were not worth emulating. In my maintenance work, I was presented with examples of good work and examples of bad work, but no one was there to help me distinguish between the two.

Working in teams, the inexperienced developer is working alongside the seasoned worker. The new person sees good work being created and sees not-so-good work being identified and improved. There is no better teacher than guided experience. The new developer can be productive without being disruptive — supportive while being supported.

The experienced developer also benefits from the team environment. Career development is enhanced by being able to work with specialists in various areas. If a programmer is interested in becoming a tester, he can choose to spend more time with the tester on his team. The business analyst may aspire to be a designer. The opportunity is there to observe and participate in the activities that will enhance her skills in related disciplines.

The opportunity for learning and career development exists between developers and managers as well. The manager is actively monitoring the repository to gauge the effort's status, allocating resources, and prioritizing tasks. She can take advantage of this shared resource (the repository) to view and study the elements created by the teams. This makes her more valuable to the team because she can help without taking developer's time to "come up to speed" and she has a better appreciation for the capacity and capability of the developers.

Developers looking to take on more managerial responsibilities are in a perfect position to see and participate in the management of the team and of the larger planning process. When both the managers and developers are working with the same knowledge repository, information is visible to everyone. Managers can (and should) involve developers in the planning and monitoring processes to enhance the skills and knowledge of all interested players.

The Team of One

Large teams can be problematic. What about small teams — or teams of one? The software industry has historically attracted individuals who work well alone.

The stereotypes of the developer pounding away on his keyboard in the middle of the night and gaining personal satisfaction from single-handedly saving the day are more fact than fantasy. So, what do you do with a developer who does not work well in teams? Or the generalist who can work alone effectively?

This one-person team is problematic. Our definition of work is a strand where every element is moved to a verified state. For an element to be considered verified, it must be reviewed by someone other than the author. As a manager, I have to ensure that someone reviews and approves the work of the one-person team after she has announced that the work is completed.

I find it much more time-consuming to have a reviewer come in and evaluate a strand after the developer thinks the work is complete. Teams working concurrently can keep up with what each member is doing. The team members have the background and context necessary to review each other's work before it is completed and while the author is still able to accept changes. After the product is "done," it is more vexing to hear criticism — even constructive, well-intentioned criticism.

Can one-person teams be effective? Yes. But I try to develop an environment in which individual efforts are replaced with collaborative work. You will have succeeded when all developers prefer working with colleagues and feel that working on an assignment alone is done only as a last resort.

While one-person teams are sub-optimal, that does not mean that developers must always work in teams. It is important to allow space and time for individuals to think and ponder. It is OK for a team member to stop working on his part of a task and take time to work alone.

To do great work a man must be very idle as well as very industrious.

—Samuel Butler II

Multi-Team Efforts

As a manager, I have to coordinate teams of teams. My experience teaches me several things:

- Most development efforts are larger than one person or one team.
- Multidisciplinary teams are the best way of organizing software development efforts.
- Self-directed teams can be very effective in creating high-quality products.
- There is a limit to the size of any individual team.

These truisms impose certain constraints. If team sizes are fixed (three to seven developers) and one team cannot reasonably be expected to perform an infinite amount of work, to get a lot of work done requires many small teams. Each new team added to a development effort requires a degree of coordination between teams (this is work in addition to the internal team

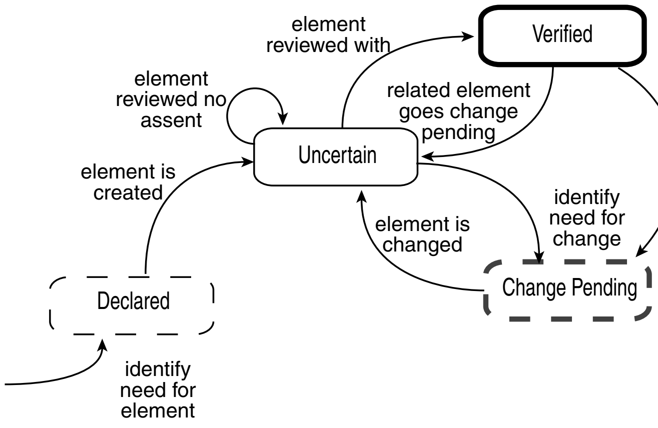


Exhibit 5 Element State Model

effort). To keep the amount of inter-team coordination work to a minimum, you use the repository to coordinate teams working concurrently on related development efforts.

It would be nice if tasks could be defined that were completely independent with no work by a team on a strand of elements affecting a strand being developed by another team. I have never seen such a thing. Everything is deeply interrelated. And as we strive for high levels of reuse and integration, our applications become more and more interconnected. The saving grace is that one team does not have to know everything about the workings of another team. They only need to be aware of each other when one team moves an element from its “Verified” state to either a “Change Pending” or “Uncertain” state (see Exhibit 5).

Visualize two teams working on strands that related to the inventory pricing. The Blue Team is enhancing the pricing function and the Green Team is adding attributes to an inventory database. The Green Team discovers that a previously built and verified Object Design should be changed in order to complete work on their task. The Object Design in question is changed from “Verified” to “Change Pending.” This immediately changes all related elements in the repository to “Uncertain.” The proposed change might have an effect on any associated element. Some developer will have to exert energy to check these associated elements for any ill effects the change may cause. If no other team is working on a strand containing elements associated with the Object Design, the Green Team is free to proceed without concern. However, if the Object Design element, or any element associated with it, is part of the strand in another team’s task, the teams need to coordinate their efforts.

The Blue Team has included that particular Object Design in its strand. No changes were immediately apparent to the Blue Team when they started their work, but they were adding modules that used the Object Design, so the Object Design is listed as part of their task definition.

The change planned by the Green Team is communicated to the Blue Team. At this point, the teams have an obligation to coordinate their efforts.

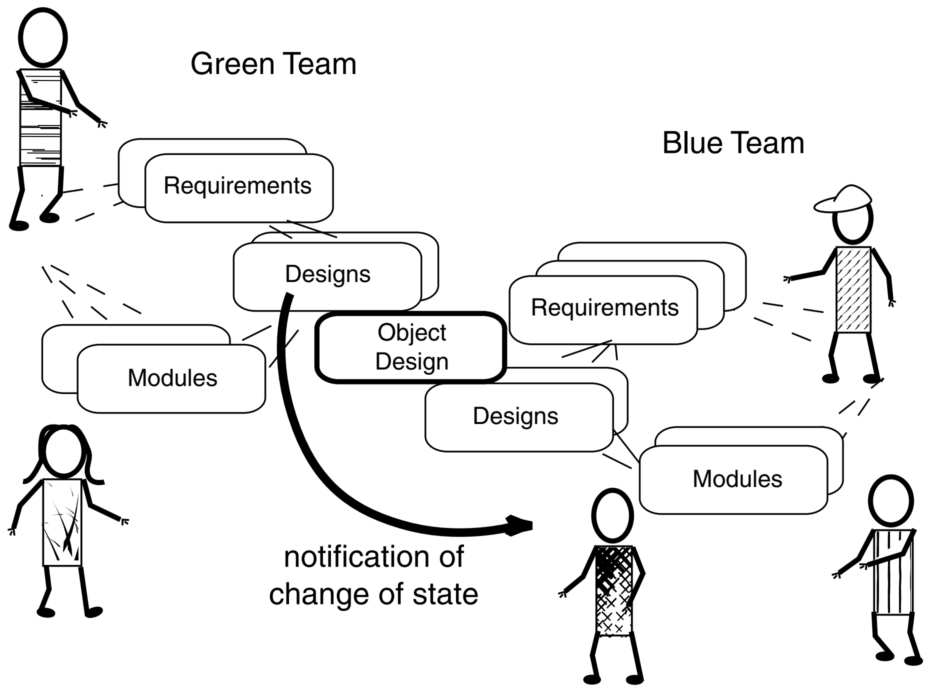


Exhibit 6 Shared Element as Bridge between Teams

This also means that the Blue Team must be part of the review that will move the new definition of the Object Design from "Uncertain" to "Verified" (see Exhibit 6).

As a manager, I try to define tasks with as little coupling as possible. The coordination may have been easier if the two tasks were assigned in sequence. I could have assigned the Green Team's task first and scheduled the Blue Team's work to begin after the Green Team was done. This way, the Blue Team would begin their work on a newly refined Object Design. On the other hand, the insight from the Blue Team's perspective may be useful to the refinement of the Object Design, and after the Blue Team starts work, they might identify a desirable change to the Object Design requiring additional work.

The fact is, one does not know which approach will require less work or be less inconvenient. It would be a waste of time to try to figure it out. Delaying needed work on the off-chance that some part of the application may change is always the wrong decision. We cannot know, but we can build an environment in which overlaps and potential dissonance are easily identified and teams are rewarded for keeping the repository in a consistent state (meaning: all elements are in a "Verified" state).

Motivating by Rewarding Consistent Work

One way to lead developers toward collaborative development is to reward people for building product that is complete, internally consistent, and verified.

By "reward" I am not talking about monetary reward. I have doubts over the long-term effectiveness of bonuses and cash awards. We tend to be motivated by expectations from our peers and from our supervisors, as well as the expectations we set for ourselves. Expectations are best communicated through regular, short, direct feedback.

Think about the Blue Team. Team members are working on the enhancement of the Pricing Function. They are assuming that the Object Design is "solid" and have begun to build elements necessary to complete their task. But suddenly they get a message saying that the Green Team is proposing a change to the Object Design. One would predict that the Blue Team might be a little irritated. But as the manager, you have the opportunity to exacerbate the situation or capitalize on it.

The goal is to build effective software solutions — continuously. That means maintaining the repository in a state where all elements are accurate, complete, and internally consistent. The wrong message to give the Blue Team is:

> "OK — Sure, you have more work to do, but you still have to be done by the end of day tomorrow."

This punishes the team for working with the Green Team, creating a path of least resistance to leave the repository in an inconstant state. This type of message rewards developers for being isolated and rewards the individual effort — although the individual effort has a detrimental effort in the larger context.

A more constructive message is:

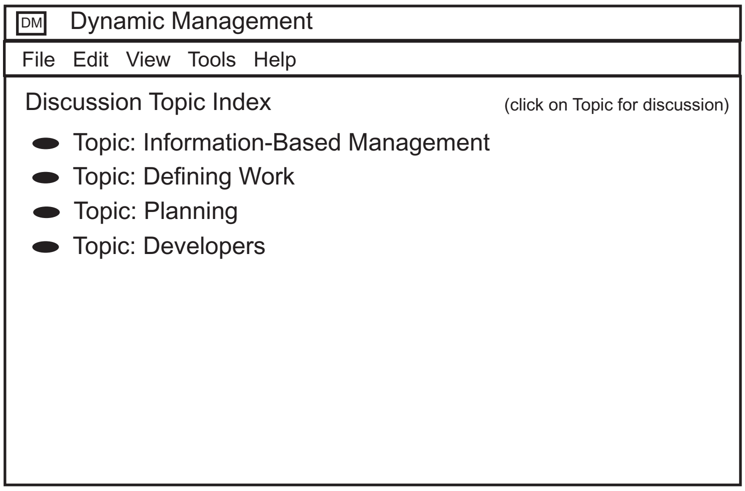
> "Check with the Green Team and help define the Object Design. Let me know how this changes your task definition."

All teams will identify the need to change elements you did not anticipate. The expectation to establish is that "care and feeding" of the knowledge about the application is far more important than maintaining the original definition of a task. All managers and developers must get into the habit of focusing on the repository and the real knowledge being created. The plans and task definitions are aids to the coordination of resources and monitoring of progress, but the plans and task definitions are not the product adding value to the user.

> They [programmers] often seem to think that their primary function is to invent clever new algorithms, rather than to perform useful work.
>
> **—Ed Yourdon**

Applying Dynamic Management

Identify a small group of developers who, in your opinion, work well together. Rearrange their office space to provide them with a dedicated work area (e.g.,



Discussion Topic Index (click on Topic for discussion)

- Topic: Information-Based Management
- Topic: Defining Work
- Topic: Planning
- Topic: Developers

Exhibit 7 Online Discussion Index

a conference space or common area outside their offices). Identify a moderately-sized maintenance or enhancement task and assign the work to the team (rather than to an individual developer). Ask the team to do everything necessary to deliver a complete, internally consistent, and reviewed product. Ask them to track the elements they build and change. After the work is complete, analyze the work the team accomplished and the team itself.

To analyze the work the team accomplished, compare the list of elements tracked by the team with the list of elements drafted as part of the "Applying dynamic management" exercises in Chapters 1 through 3. Discuss similarities and differences with the team. Ask them to identify the elements and associations they found most useful.

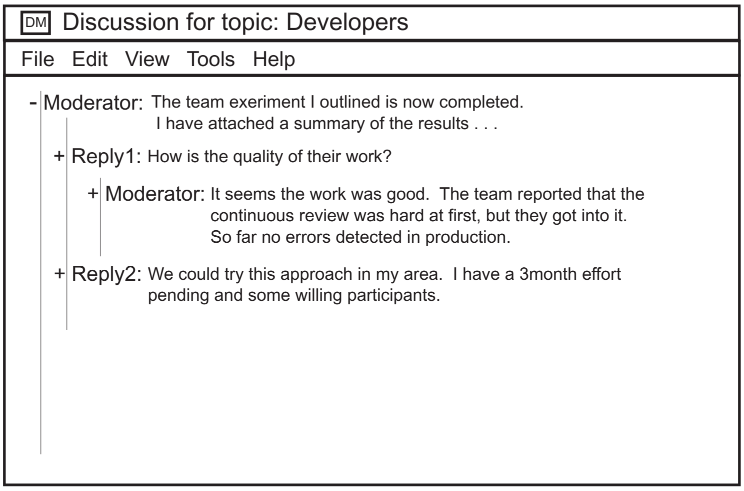
To analyze the team itself, discuss the team approach to application development with the developers. Did every developer feel that the group was functioning well? Did they resolve interpersonal issues effectively? Did they allocate work based on the skills and interests of the individuals? Did any of the developers find it difficult to critically review other team member's work product? (See Exhibit 7.)

Add a topic to your online discussion group and attach a summary of your team's experience (see Exhibit 8). Invite members of your discussion group to review and comment on the results. Explore ways to expand the experiment to other development efforts.

Chapter 4 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

The developer's conference was notable for two reasons: (1) the developers from headquarters and the Midwest left with a sense that they were all in the



DM Discussion for topic: Developers

File Edit View Tools Help

- Moderator: The team exeriment I outlined is now completed. I have attached a summary of the results . . .
  - + Reply1: How is the quality of their work?
    - + Moderator: It seems the work was good. The team reported that the continuous review was hard at first, but they got into it. So far no errors detected in production.
  - + Reply2: We could try this approach in my area. I have a 3month effort pending and some willing participants.

Exhibit 8 Beginning Dialogue

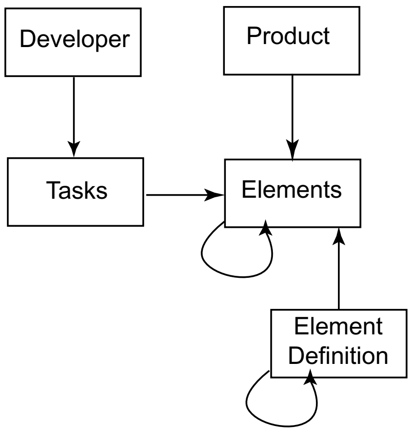


Exhibit 9 Initial Information Model

same boat; and (2) the developers knew that their managers were interested in improving the way software was built at DSM and were willing to listen to their ideas. After the now-famous conference room discussion, many hallway discussions occurred. By Thursday, there was a consensus that however they redefined “process” and “phase,” they needed a common repository to store their “stuff.” The exact definition of “stuff” was elusive, but a shared memory was going to be critical to their success. In the minds of the West Coast developers, this was visualized as a great knowledge base where all the project deliverables were stored. In the minds of the Midwest developers, this was visualized as a database storing their collective memory.

As they talked, a simple model emerged (see Exhibit 9). They all agreed that the products they built consisted of many different kinds of elements (e.g., pages of requirements, pieces of designs, code, libraries, user manuals,

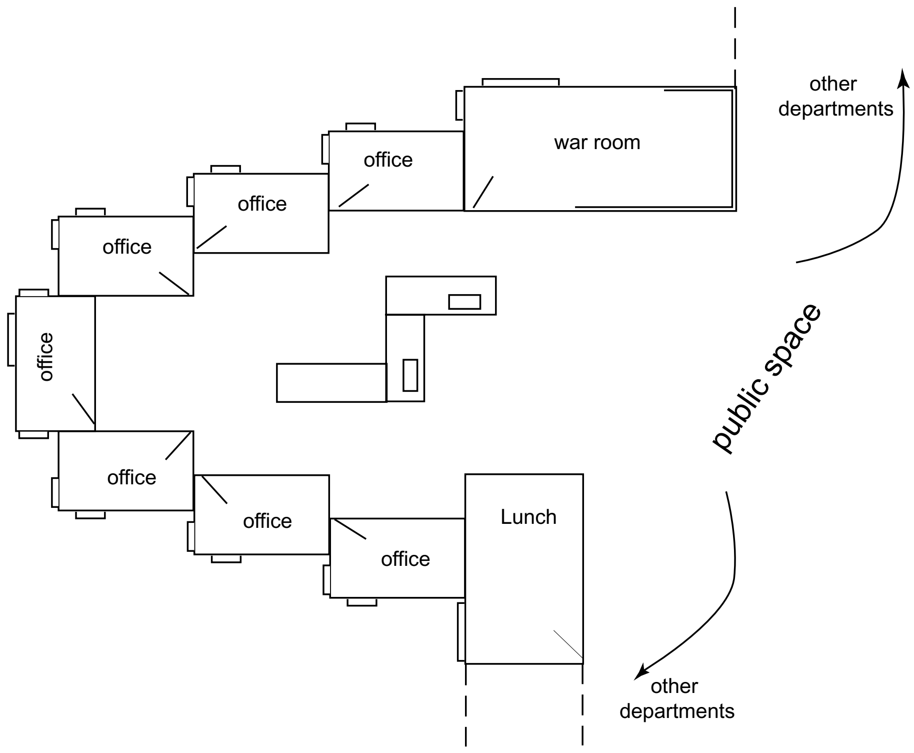


Exhibit 10 Midwest Development Work Area

database tables, interface designs, test cases, etc.). They all agreed that they worked by identifying some objective (i.e., a task) and building the elements as they work toward that objective. However, interesting insights began to emerge as they discussed the elements themselves. Elements were connected — not to other elements of the same type, but to elements of different types that described the same feature or subsystem. The "cohesive chunks of the product" that the Midwest manager drew in the conference room were strands of pieces of knowledge about a function.

Before leaving the conference on Friday, the developers had added a notation to their informal model to represent the associations among elements. They also decided there had to be some agreement as to what those elements were and what associates were typical, so an entity called "element definition" was added to the model.

The action item accepted by every developer was to begin defining each model in greater detail and to start populating the models with actual instances of elements and element definitions. No one was sure where this would lead, but all agreed it would be an interesting journey.

Over the course of the next year, the CIO deliberately defined development efforts involving both West Coast and Midwest developers (see Exhibit 10). Many of developers at DSM headquarters spent time at the Midwest facility. Part of every project plan was the enhancement of the repository model and improvement of the development practices using it.

The developers visiting from the West Coast got a real appreciation for the connection between physical surroundings and philosophy. Each member of the development staff at Midwest had his own office. There were a couple of workstations set up in the common area where users and developers had frequent informal meetings to discuss the latest features and discuss plans for the next round of improvements. But what intrigued the West Coast developers most was the time spent in the "war room." The war room was a conference room dedicated to the development group (i.e., they had exclusive use of the space). It was used as a command center. The walls and whiteboards were thick with diagrams and notes and code printouts. Several times a day, groups of developers would confer about the current project. One of the Midwest developers commented that it was great to be able to walk into the war room and immediately have a sense of the group's status.

By contrast, the Midwest developers spending time at the West Coast facility noted that the partitioned layout seemed to dictate a partitioned approach to work. With no space to exchange information informally, the developers had to pass information from person to person in a more structured way.

The developers set up an online conference with a discussion section to share their ideas on the repository and how it should be used. Within six months, they had put together some prototypes of a database system for the repository and were updating it with their element definitions. As they refined or added definitions to the repository model, the developers noted many common element definitions. Each shop included elements such as:

- Test cases associated with units of code
- Requirements associated with object/entities
- Object/entities associated with database tables

There was disagreement as to what constituted a unit of code. The West Coast used mostly C; Midwest used C++ and some Java. There were differences in the format and size of the "requirements statements." The West Coast used a variation on the formal DSM-DM with defined event and data definitions, and the Midwest requirements read more like narratives. The West Coast documented business rules associated with data object definitions while the Midwest recorded their policy rules with the requirement narratives.

The greatest disagreement involved the definition of "design." Both groups used different methods and models to represent design decisions. The West Coast made use of state models for interface designs while Midwest used less formal navigation diagrams. Both used class diagrams but different graphic notations.

Vocabulary was also a difficult issue. Each group had its own definitions for common words. "Object" and "Entity" were synonyms at Midwest but they had distinct definitions at DSM headquarters. Midwest used "use cases," which were very similar to what the West Coast called "event definitions." The manager at Midwest and the CIO often got together and declared company definitions when consensus was not reached quickly.

The results of the effort were rather impressive. While the development efforts initiated since the conference were small (i.e., none larger than four-month duration with four to eight developers), the work being completed was good. The users liked the fact that real solutions were occurring often, with an apparent improvement in overall quality. The developers loved building good applications as well as building effective development techniques. They also liked the idea that they were significant players in defining the methodology although no one used the word "methodology." One developer said it felt like they were creating a culture — not a method.

There were problems. The biggest was with West Coast managers, who were feeling uneasy about the lack of control. They complained to the CIO that there was no good way of measuring productivity and monitoring progress. "Users ask for something and the developers give it to them," complained one manager. "We need a way to bring back some planning into the equation." The CIO agreed that the managers needed to get more involved in the process of defining development methods. And while he did not say anything, the CIO thought to himself that giving the users what they want is not entirely bad.

Chapter 5

Monitoring Productivity…

Or Getting Better All the Time

I have made reference to productivity measurements in previous chapters. This chapter defines how the repository is used to generate productivity figures. These numbers are useful in monitoring progress, evaluating effects of improvement efforts, and projecting estimates of proposed work.

> Get your facts first, and then you can distort them as much as you please.
>
> **—Mark Twain**

Measuring Work Done

Chapter 2 equated work with chunks of information. Elements added to the repository are evidence of work done. Chapter 3 initiated a discussion of productivity with the example of the Blue Team and Clayton. Now we can dig deeper into the issue of measuring the amount of work done by developers and teams in a continuous development effort.

As Clayton and the Blue Team are working, the effects of their efforts are reflected in changes in the development repository. In the information model in Appendix D, these changes are referred to as "improvements." These improvements are reflected in the repository as elements being transformed from state to state. That is, a developer building an initial version of an element moves the element from "declared" to "uncertain." An element being reviewed and approved moves the element from "uncertain" to "verified." As the developer works on a task, new elements are created, existing elements are changed, and elements are verified or reviewed.

Exhibit 1 Development Plan

Task ID	*Focus Element*	*Element Count*	*Description*	*Priority/ Sequence*	*Assigned*
A	Customer Object Design	20	Add customer order status	Done	Blue Team
B	Customer Object Design	12	Add customer credit query	Done	Clayton
C	Product Object Design	23	Enhanced product pricing formula	1	Blue Team
D	Product Object Design	8	Replace inventory valuation method	2	Clayton
E	Product Object Design	17	Add product inquiry	3	?

Exhibit 2 Examples of Association Definitions

Association →			*← Association*		
Element	*Meaning*	*Avg. Freq.*	*Avg. Freq.*	*Meaning*	*Element*
Requirement	suggests data defined as	2	5	defines data suggested in	Object Design
Object Design	is stored as	3	1	implements	Relational Table
Relational Table	is accessed by	3	2	accesses	Java Class
Java Class	implements	1	1	is implemented as	Object Design
Java Class	is exercised by	4	1	exercises	Test Suite

By way of example, I will describe the way Task C is accomplished by the Blue Team (declared in Exhibit 1). The plan estimated that the task involved 23 different elements. The Blue Team's objective is to enhance the product pricing formula.

The team finds the current formula described by a strand consisting of the Requirements, one Object Design, a related Relational Database Table, one Java Class based on the Object Design (methods of the class access the database table), and three Test Suites. This strand is typically based on the association definitions depicted in Exhibit 2. The frequency declared in the table should be computed from the current repository. It means that Requirements in the repository are associated with a average of two Object Designs. Each Object Design is associated with about two Relational Tables. On average, each Relational Table is accessed by three Java Classes, and there are typically four Test Suites built for each Java Class. So, if the Blue Team determines that Task C will require one additional Object Design, we can project that the team will also have to build (or reuse) three Relational Database Tables, three Java Classes, and four Test Suites.

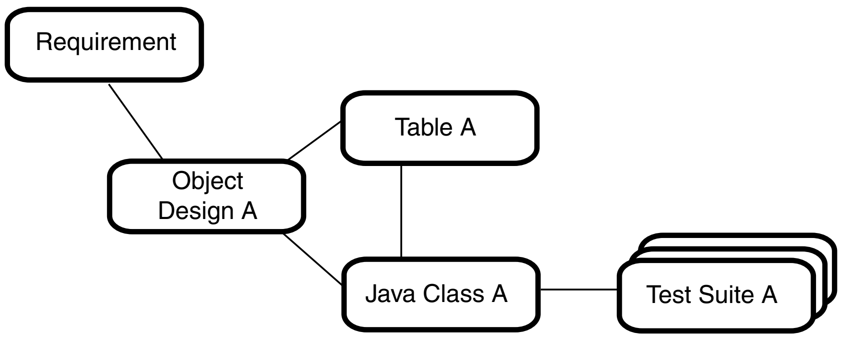


Exhibit 3 Before Task C

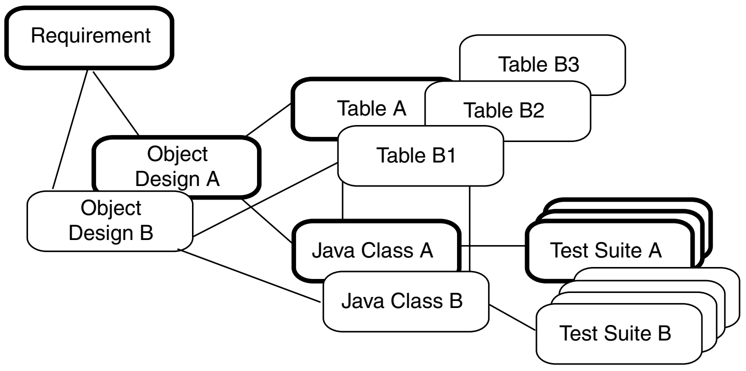


Exhibit 4 After Task C (Projected)

Exhibits 3 and 4 graphically predict the projected work of the Blue Team in their effort to complete Task C. Currently, the repository shows the pricing function is described by its Requirement, one Object Design, one database table, one Java Class, and three Test Suites. This strand is smaller than average. Based on the association definitions in Exhibit 2, the average requirement is associated with two Object Designs, three Relational Tables, three Java Classes, and four Test Suites. The frequencies are averages. The repository would have larger strands.

The element and association definitions determine the estimated work requirements for a task.

Exhibit 4 also depicts the repository after the Blue Team completes Task C. If the team is correct in projecting the need for one more Object Design, and the average frequencies from the repository are accurate, the team's work is defined by three Table definitions, one Java Class, and four Test Suites.

At this point, we do not know if the existing elements will need to be changed or if some of the projected elements can be defined by reusing

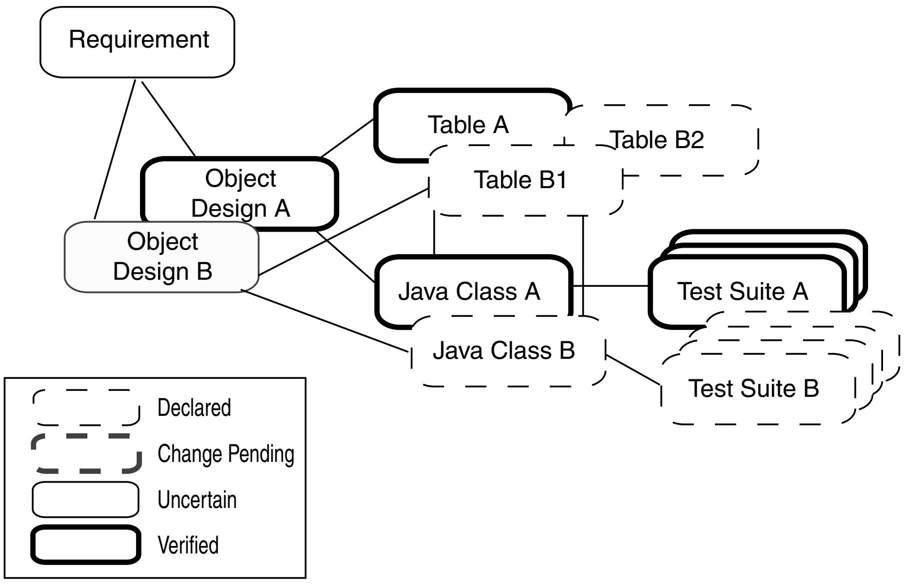


Exhibit 5 After Day 1

existing elements. But in either case, the team will have to exert work to move the repository from its current consistent state to a new (better) consistent state.

Measuring What Has Changed

The repository in Exhibit 5 reflects the work performed by the Blue Team after the first day. The team created the new Object Design and discussed the characteristics of the associated elements. They decided that the Object Design would require two database Tables (rather than the average of three). They determined that the new Object Design would be implemented as one Java Class and that it was likely to require four Test Suites to exercise it effectively. The work performed by the team can be summarized by listing the transitions made to each element (see Exhibit 6). The transitions can be called "Improvements" because the work, insight, and creativity of the team results in improvement or added value to the repository.

The new Object Design is built, so its state moves from "Declared" to "Uncertain." The "Uncertain" state means the element exists and the developers feel it is complete, but it has yet to be reviewed for consistency with associated elements. The "Declared" state means that the developers know the element is needed, but it does not yet exist (or has not been completed).

Work went into each of these improvements. I do not know, nor do I care, if Improvement 3 took more time than Improvement 8. Over time, things even out.

After the second day of work, the Blue Team had made more progress and that good effort is reflected in the repository (see Exhibit 7).

Exhibit 6 Day 1 Improvement Log

	Element	*Improvement*
1	Object Design B	Declared
2	Object Design B	Uncertain
3	Table B1	Declared
4	Table B2	Declared
5	Java Class B	Declared
6	Test Suite B1	Declared
7	Test Suite B2	Declared
8	Test Suite B3	Declared
9	Test Suite B4	Declared

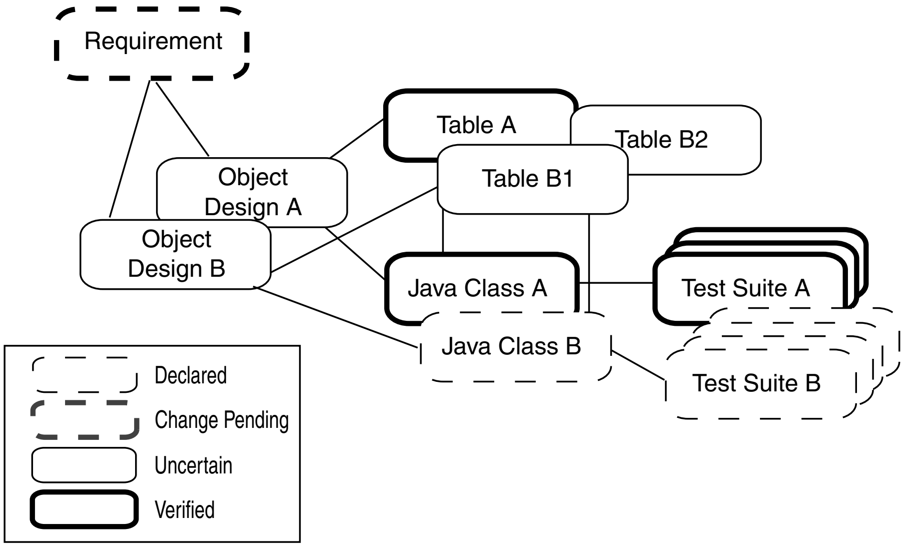


Exhibit 7 After Day 2

Exhibit 8 Day 2 Improvement Log

	Element	*Improvement*
10	Object Design B	Uncertain
11	Table B1	Uncertain
12	Table B2	Uncertain
13	Requirement	Change Pending

Developers working on a database defined the Relational Tables that will store attributes of the new Object. In completing their work (see Exhibit 8), the developers mark the original Requirement as “Change Pending.” The need to update our knowledge of the requirement was clear from the beginning.

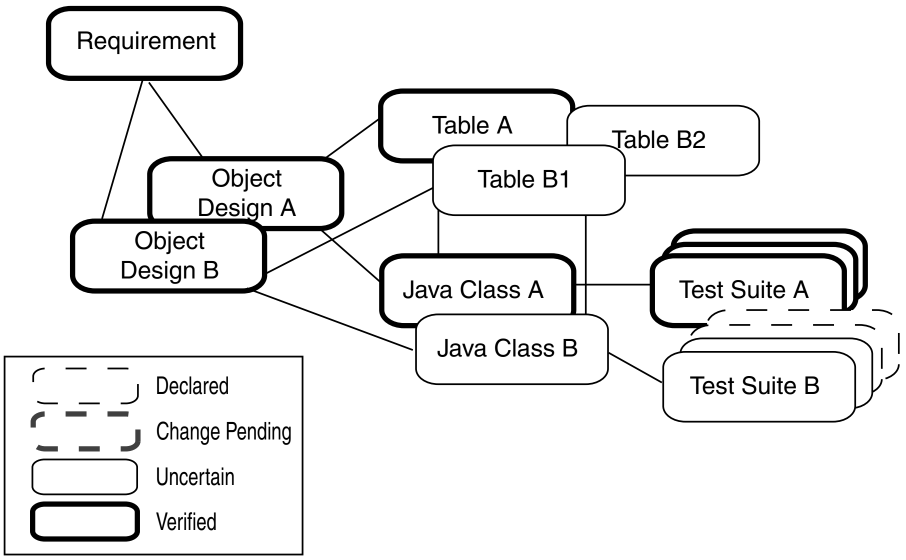


Exhibit 9 After Day 3

Exhibit 10 Day 3 Improvement Log

	Element	*Improvement*
14	Object Design B	Verified
15	Requirement	Verified
16	Java Class	Uncertain
17	Test Suite B1	Uncertain
18	Test Suite B2	Uncertain

The original objective described in the Task was to enhance the product pricing formula. Enhancing the system to consider additional information (i.e., the new object) must be reflected in the business knowledge retained in the repository. The team started this effort today and are reasonably sure what they have changed is correct, but the day ended without consensus. So, the Requirements element remains in a state of "Change Pending." The Object Design A is also in an "Uncertain" state because its associated Requirement is being changed. Object Design A might not need any adjustment, but it will have to be reviewed before this part of the product is verified. The "Uncertain" designation records that work must be performed on this element.

After Day 3 (see Exhibit 9), the work is nearly complete, but changes in the repository are binary. There is no such thing as an improvement that is 75 percent complete. The repository states that the team has reviewed the changes to the Requirement element and have concurred that the new version of the requirement is correct and consistent with all associated elements (see Exhibit 10). Object Design B is also verified against its associated Requirement, Tables, and Java Class. The Java Class and two of the associated Test Suites

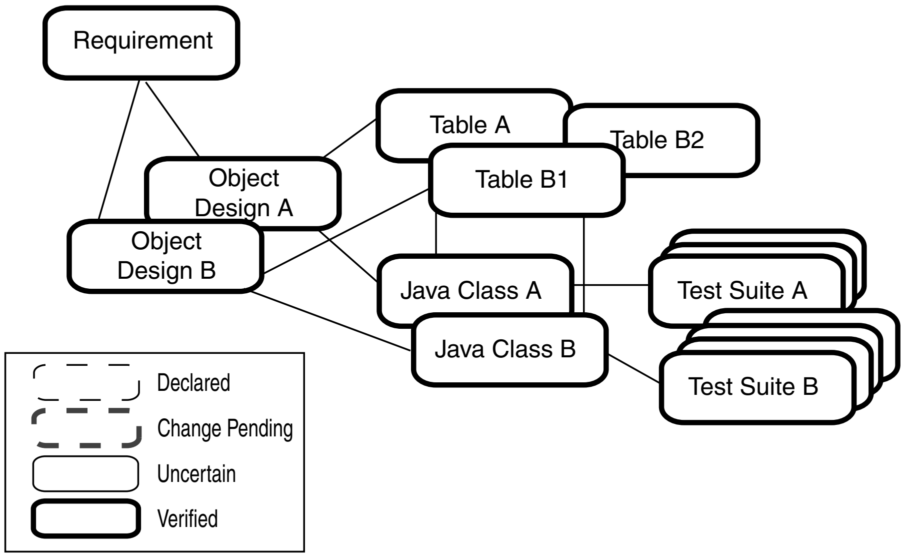


Exhibit 11 Day 4

Exhibit 12 Day 4 Improvement Log

	Element	*Improvement*
19	Table B1	Verified
20	Table B2	Verified
21	Test Suite B3	Uncertain
22	Test Suite B4	Uncertain
23	Java Class	Verified
24	Test Suite B1	Verified
25	Test Suite B2	Verified
26	Test Suite B3	Verified
27	Test Suite B4	Verified

are in the "Uncertain" state. The team has not reviewed the Tables (B1 and B2) and associated Java Class, so these elements remain "Uncertain." Two Test Suites are still being built, so they remain "Declared."

On the morning of the fourth day, the remaining Test Suites are completed, moving them to the "Uncertain" state. The team performs the remaining activities of reviewing the Tables against the Java Class, and the Java Class against the Test Suites. Once all have concurred that the elements are correct and consistent, Task C is done (see Exhibits 11 and 12).

The Blue Team has been working on the task for four days. They have accomplished an objective of enhancing the product pricing formula that was originally described with seven elements. It now consists of 15 elements. The repository (i.e., knowledge) is now a consistent state and ready for future work. The organization has "documented" the work in a way that records the knowledge necessary for future enhancements.

Actual work is defined by the new and changed (and deleted) elements in the repository. The work is complete when the repository reflects all we know about the current state of the product.

Determining Work's Cost

The Blue Team consisted of four developers, each working 3.5 on Task C. Each team member was on the job for about eight hours each day (four hours on Day 4). A total of 192 person-hours were dedicated to the task. This time can be allocated to the various elements simply by distributing the hours evenly across the improvements and summing by element. The changes in the repository suggested 27 improvements or a little more than seven person-hours (7.11) for each. I can distribute the precise fraction, but I prefer to simply allocate whole numbers. The decision to allocate seven hours (or eight hours) is simply a guess as to the relative size and complexity of the element. A more accurate allocation can be implemented by weighting the hours by the Weight Units of the Element Definitions (refer to Chapter 3). For this example, I allocate the hours as shown in Exhibit 13. Summing the allocated hours by element results in an approximation of the hours each element consumed during Task C (see Exhibit 14).

I am not interested in when work was done, in which order, or by whom. The team is responsible for meeting the task's objectives. The task is accomplished by adding value to the repository. The value is tracked by the changes of state of the related elements. There is no attempt to factor in lunch breaks, time spent looking up articles on developer-oriented Web sites, and talking about yesterday's ballgame. If team members spent 10 percent of their time on these social activities while working on Task C, they would probably spend 10 percent of their time on similar social activities while working on a future task.

These numbers are valuable for future planning. As new tasks are defined, the historical data is used to project how long the task will take. A team of similar composition and skills working in the same domain will probably work at a similar rate.

The actual cost is the total resource charged to the development effort. That cost can be allocated to the elements listed in the improvement log.

Demanding Enhanced Value

Looking back to Chapter 3, it was estimated that the task would involve 23 different elements and take approximately 3.5 days (given that the Blue Team has been working at a rate of six to seven elements per day). Task C actually involved 15 elements, of which nine were new or changed. Their productivity

Exhibit 13 Hours Allocation

	Element	*Improvement*	*Hours*
1	Object Design B	Declared	7
2	Object Design B	Uncertain	8
3	Table B1	Declared	7
4	Table B2	Declared	7
5	Java Class B	Declared	7
6	Test Suite B1	Declared	7
7	Test Suite B2	Declared	7
8	Test Suite B3	Declared	7
9	Test Suite B4	Declared	7
10	Object Design B	Uncertain	7
11	Table B1	Uncertain	7
12	Table B2	Uncertain	7
13	Requirement	Change Pending	7
14	Object Design B	Verified	7
15	Requirement	Verified	7
16	Java Class B	Uncertain	8
17	Test Suite B1	Uncertain	8
18	Test Suite B2	Uncertain	7
19	Table B1	Verified	7
20	Table B2	Verified	7
21	Test Suite B3	Uncertain	7
22	Test Suite B4	Uncertain	7
23	Java Class B	Verified	7
24	Test Suite B1	Verified	7
25	Test Suite B2	Verified	7
26	Test Suite B3	Verified	7
27	Test Suite B4	Verified	7

Exhibit 14 Hours/Element Summary

Element	*Hours*
Requirement	14
Object Design B	29
Table B1	21
Table B2	21
Java Class B	22
Test Suite B1	22
Test Suite B2	21
Test Suite B3	21
Test Suite B4	21

Exhibit 15 Task: XYZ Enhancement

Element	*Transition*	*Comment*

rate for the four-day task is three to four elements per workday. Does this mean the team's productivity is dropping and I should crack the whip or disband the team? Probably not. This task's information is one of many gauges you should use in monitoring the development effort.

The first thing to communicate to the team is: "Good work! Thank you for achieving your goal of adding value to the services we provide to the enterprise." The goal is not higher productivity, it is greater value. The Blue Team's objective is to enhance the product pricing formula and, if that is accomplished, the environment is functioning well.

> Two orchestras play the same symphony. The one that finishes first is not necessarily the best.

If I see productivity numbers dropping over time, I will look for causes and plan actions aimed at enhancing performance. Teams can always use more skill, better tools, and closer working relationships with users and policy makers. If I see productivity numbers improving, I will look to see if the change can be attributed to process improvement efforts recently initiated.

I would also check to verify that the "improved" numbers are not a symptom of declining quality. The enterprise is not served by a development group that is producing lots of bad stuff. The enterprise is served by a development group that produces valuable solutions day after day, month after month — regularly and consistently.

Applying Dynamic Management

Discuss the ideas in this chapter with the development team defined at the end of Chapter 4. Identify the work products produced by the team while working on the task. Identify the changes produced by the team that indicate progress. Can you use those changes to monitor the progress of the task?

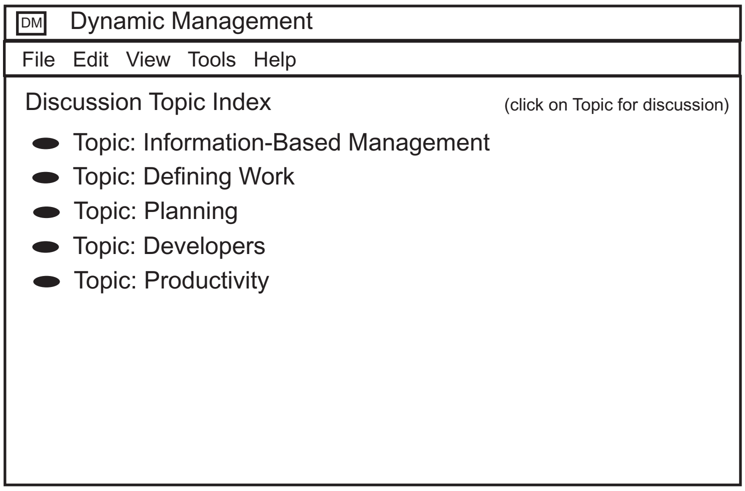
Discuss the idea of an improvement log.

Build a representation of such a log for the last task performed by the team (see Exhibit 15).

Do the developers feel the log can be integrated into their development environment (i.e., have log entries been automatically created by patches to their development tools)?

Discuss the idea of allocating time to the element via the improvement log.

Do the developers feel this information is representative of the value they add to the applications?



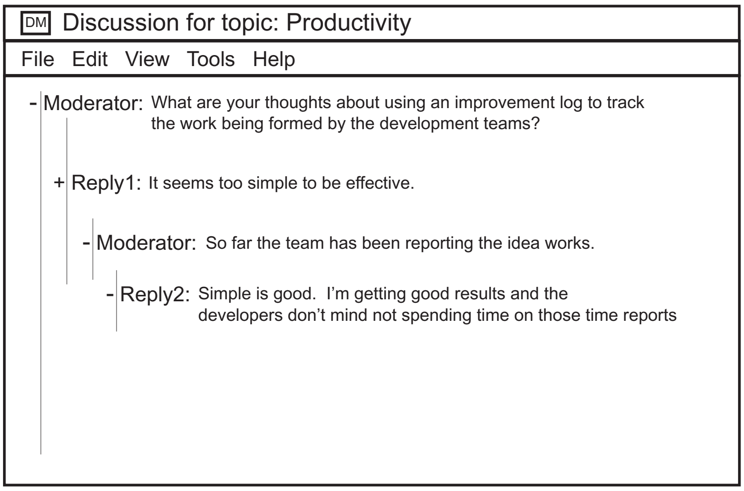
DM Dynamic Management

File Edit View Tools Help

Discussion Topic Index (click on Topic for discussion)

- Topic: Information-Based Management
- Topic: Defining Work
- Topic: Planning
- Topic: Developers
- Topic: Productivity

Exhibit 16 Online Discussion Index



DM Discussion for topic: Productivity

File Edit View Tools Help

- Moderator: What are your thoughts about using an improvement log to track the work being formed by the development teams?
  + Reply1: It seems too simple to be effective.
    - Moderator: So far the team has been reporting the idea works.
      - Reply2: Simple is good. I'm getting good results and the developers don't mind not spending time on those time reports

Exhibit 17 Beginning Dialogue

By watching changes in the repository, you reduce or eliminate the need to require status reports from individual developers. This allows you to focus on monitoring and controlling the development effort, while your developers are focused on creating effective solutions (see Exhibit 15).

Add a topic to your online discussion group and explore the idea of using an improvement log to track the states of the elements (see Exhibit 17).

Chapter 5 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

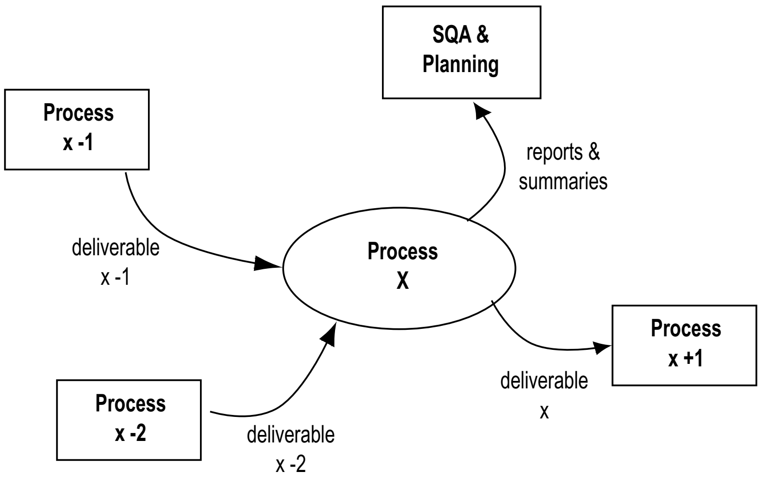


Exhibit 18 Old Generic Development Process

The development group of DSM first became interested in the CMM about 13 years ago. The first articles and books published that explored the idea of a set of observable attributes of a competent organization caught the imagination of many members of DSM's management. DSM never considered formal appraisal, but did invest a great deal of time and energy in modifying its methodology (DSM-DM) to reflect the Carnegie Mellon University Software Engineering Institute's maturity model.

The DSM managers were particularly interested in developing and implementing planning and measurement techniques. There was a sense of optimism that the software development process could be manageable and predictable, if only the work could be performed in a consistent manner and deliverables could be analyzed and measured. The DSM methodology provided a well-defined framework of the development process (see Exhibit 18). Each task had pre-conditions (i.e., completed deliverables from previous tasks) and a clear objective (i.e., the creation of a deliverable used in the next task).

As DSM gained more experience with measurement activities, the managers allocated more of their resources to the quality assurance process. Several of the analysts were given the job of developing techniques for monitoring the development process and providing measurements of quality and productivity.

The developers were asked to provide more detailed information about the tasks they were performing. Programmers were asked to provide reports on the complexity of their programs. Designers had to include in their status reports the counts of interface complexity and evaluations of the degree of class reuse. Deliverables from database design tasks had to include measures of table normalization and transaction optimization.

While the information gathered from the measurement program was useful, the additional requirement of reports and summaries was adding to the workload of the developers. The extra work was minimal when the development task was small. For larger efforts, or for projects that had fallen behind schedule, the developers complained that the imposition was hindering their progress. It was not surprising that the developers took little care in the accuracy of the reports and summaries the developers prepared for the SQA (software quality assurance) and planning analysts. After all, the real work had to get done and these reports were not helping them get the product out the door.

The analysts working on the measurements were able to produce impressive projections. When the DSM-SISS project was being planned, the measurements from past development efforts formed the basis for estimates of time and resource requirements.

After that project's failure, managers and executives of DSM wanted to know how the estimates could be so far off. The analysts obtained information about the Midwest product and created a set of numbers based on its actual features and code. They discovered that the Midwest product was twice the size of the product described by the initial requirements statement of the DSM-SISS (two and a half times the number of function points). They reported that the features the marketing group were trying to add to the project during the first six months would have doubled the estimate for the project.

The SQA analysts concluded their report by saying that the original estimates were correct given the projected size of the product, but that the requirements statements were not accurate. The Director of Applications Development asserted, with a great deal of frustration, that the development team could not have known the full extent of the system's requirements. That knowledge was not available until they and the regional managers had a chance to refine the customer's view of the product.

One of the post-mortem meetings concluded with the CEO saying:

> "So, it appears we can accurately measure our software products after they are finished. We can accurately say what a project should have cost us if we had known everything at the beginning of the project that we know at the end. But we can't know at the beginning of a project what a project will require until the end." He shook his head and said, "Work on it."

The push for incremental, repository-based development had many of the DSM managers and SQA analysts concerned. They expressed their discomfort to the CIO by saying that the developers seemed to be acting more like hackers than professional developers. They reported that the code being produced seemed to be of good quality (there was a 64 percent reduction in the number of defects reported in the first six weeks of operations). And they liked the fact that the developers were actively reviewing each other's products. However, there was no formal measurement like they had when they were following the DSM-DM.

The CIO could see that the managers were unclear about how they fit into the scheme of things. Was this process going to put them out of a job? What was the connection between the work that the developers were doing and the strategic planning process? How should decisions about resource allocation and project prioritization be made?

The managers and the CIO began a concerted effort to rethink how planning and measurement should best be accomplished given the good work that was coming out of the ranks of the developers. They started by admitting that the accuracy of the data they had been gathering could be better. The CIO asked if there was a way of gathering information that did not require the developers to do extra work. He then suggested that the repositories being developed seemed like a gold mine of useful information.

The SQA analysts and manager began paying closer attention to the online conference and the discussion of the project repository. They began to formulate a model that shifted focus from the individual deliverables produced in processes defined in the methodology to the changing state of the repository itself. They noted that the developer's tasks could potentially add/change/delete many different types of information stored in the repository. For planning purposes, they needed to study the pattern of these changes.

Out of this analysis came a proposal to maintain an "improvement log" — a simple list of the changes made to the repository. The physical form of the repository included the company's configuration management systems, a newly implemented "pending task list," and the directories where the project documents were stored. They augmented the development environment to update the improvement log as the developers checked-in configuration items and made changes to the project directories. This allowed the information to be captured with nearly no extra work required on the part of the developers. The SQA analysts began writing scripts to query the configuration management system, the project directories and entries in the improvement log, and to subsequently produce summaries of activity with flags calling attention to unique situations.

As time went on, the concern and apprehension of the managers and SQA analysts changed to intrigue and surprising enthusiasm. One analyst commented, "We are finally measuring something real rather than regurgitating what others chose to tell us."

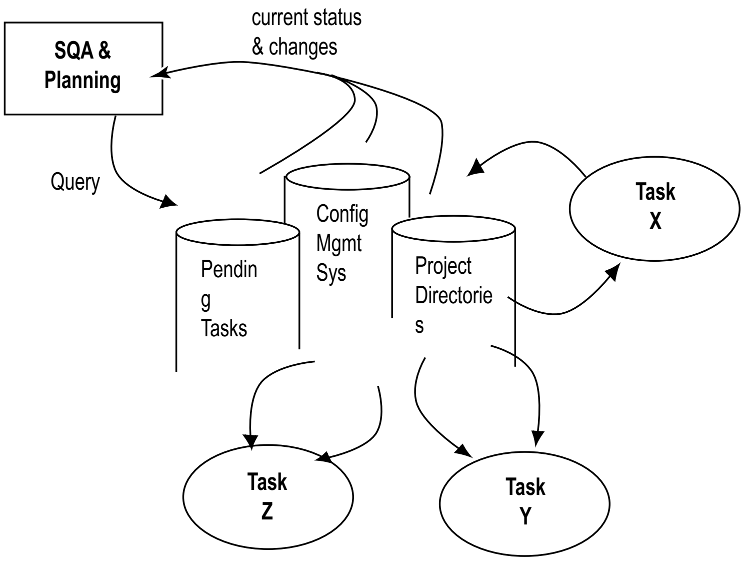


Figure 19 New Generic Development Process

Chapter 6

Strategic Framework…

Or Metadesign Integrity

A major role of managers and developers is to maintain an accurate model of how software is implemented. This model is a statement of how elements of existing technology will be employed to provide service to the enterprise. The vast majority of implementation decisions for any given function should be "generic."

> It is a profoundly erroneous truism … that we should cultivate the habit of thinking about what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them.
>
> **—Alfred North Whitehead**

The Importance of System Architecture

Every development organization creates policies (i.e., decisions made ahead of time) for how systems will be built. The system architecture of a development environment is a set of collective decisions the organization makes governing the shape of the systems it creates. This architecture is created within a spiral of decision making that runs up and down the enterprise's management structure.

Corporate-level decisions set global aspects of the architecture. As a result of these decisions, developers start the process of building the elements in the repository knowing the operating system, the technology available to the users, and the characteristics and requirements of the network (see Exhibit 1). This level of architecture is created by your buying decisions. This architecture

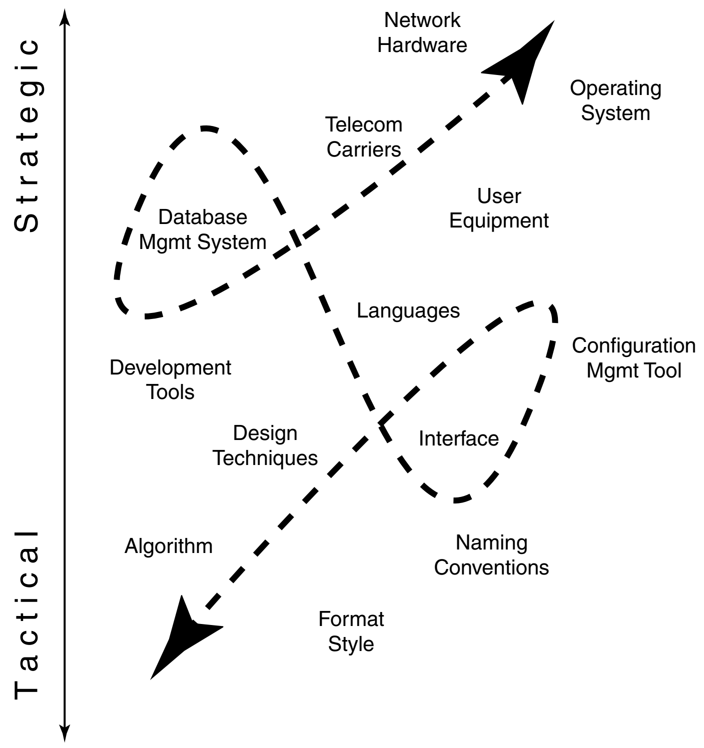


Exhibit 1 Architecture Decision Spiral

is embodied in the hardware, operating systems, networks, development tools, and utilities you buy or lease.

The decisions relating to choice of methods, type and content of training, development tools, and language are typically determined at the level of the development organization. This level of architecture is formed by the training you choose and the techniques you adopt. If you hire a large number of developers with heavy C experience and reinforce their expertise with training in advanced C, you are creating an expectation that the parts of the system requiring code will be written in C. It is not likely other languages will be considered when all evidence indicates that the implicit decision has been made: if code is needed, write it in C. If the training is conducted by a consultant advocating a state-driven design method, the environment tends to adopt a rule: when the situation calls for design, think state-driven.

Individual developers make similar decisions about interfaces, algorithms, formats, and styles. You cannot consider all variations of style while writing code, so you "standardize" on certain rules. You often have no objective rationale for any given convention; but by being true to your "style," you avoid a lot of thrashing. As soon as you know you will write a function, your style rule kicks in and says "functions are created this way." When you see you will have to write a loop construct, you pull out your standard loop from memory and simply fill in the details.

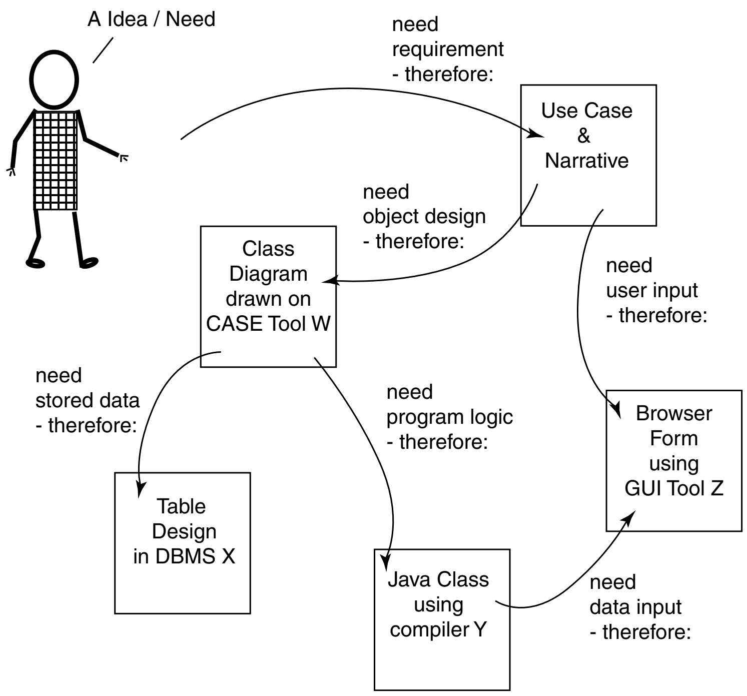


Exhibit 2 Decision Mapping

We must recognize the strong and undeniable influence that our language exerts on our ways of thinking, and in fact defines and delimits the abstract space in which we can formulate — give form to — our thoughts.

—Wirth

The sum total of these decisions defines the architecture in which all the enterprise's software is created. These decisions become patterns or rules. These rules constitute decisions made ahead of time for nearly all elements added to the repository. As the need for some information is determined, the architecture defines the information's implementation. For example, the development team recognizes the need to define the user's requirements (see Exhibit 2). The rules of the architecture map that recognition into an implementation — perhaps representing the requirements as Use Cases and supporting Narrative. As the requirements are being discussed, it becomes clear that some information needs to be defined in more detail. The rules of the architecture direct the developer to map the need into a Class Diagram drawn on the CASE tool purchased from W corporation. The CASE tool supports Java and generates initial code structures based on class information. This becomes part of the architecture. When you identify the need for program logic, the decision to generate Java Class is given. You have standardized on the Java environment from the vendor Y, so the class is compiled using that tool.

During the design of the function, the team determines the need for persistent data members of the object. The enterprise has a license for the X DBMS, so the decision of what form the stored data will take is already made. The discussion with the user was facilitated by a prototyping tool from vendor Z. When the requirements were defined, the proposed interface was outlined and the tool generated a browser form; this being part of the architecture, the decision is already made. You map your discovered need into the form and format of the architecture.

Every organization creates these tenets, if for no other reason than it is too complex, too costly, and too confusing to make these decisions for each application. The architecture forms a collective pattern and it serves many useful purposes, including:

- It allows developers to focus on applying the technology toward service to the enterprise.
- It makes it possible to target training and development efforts.
- It helps to ensure consistency across application areas.
- It increases the opportunity for reuse of existing work.

Form is liberating.

Technology Decisions

Just as software applications are dynamic and ever-changing, system architecture is also fluid. The decisions as to which elements of technology will be adopted by the enterprise require consideration from various experts at different levels of the enterprise. More strategic levels of the architecture require financial consideration, business planning, industry assessment, and expertise of related technology. More tactical levels of the architecture require information about current development practices (both inside and outside the enterprise) and knowledge of current developers skills and expertise.

The decision-making process must be isomorphic with the architecture itself.

Decisions about architecture always start with someone making a suggestion. The Chief Information Officer makes a proposal to replace our relational database management system with this new object-oriented database management system. One of the development managers suggests that Java would be a better language than the currently used C. A senior analyst raises the option of using component design techniques rather than the object-oriented design the developers are using now.

For each given proposal, the enterprise needs to consider the change to the architecture by bringing the proper expertise to bear on the decision (see

Exhibit 3 Level of Decisions

Proposal: Change DBMS

Questions/Issues	*Level*	*Expertise*
Cost of new DBMS	Strategic	Chief Financial Officer Vendor representative
Does the proposed DBMS fit with other components of the development environment? (refer to Chapter 7)	Middle	Development managers
Can new development use the new system while working with the current DBMS?	Middle	Senior developers
Will software solutions using the new DBMS be superior to those products with the current DBMS?	Tactical	Senior developers Current users of the proposed DBMS

Exhibit 4 Level of Decisions

Proposal: Use component design instead of object-oriented design

Questions/Issues	*Level*	*Expertise*
Are there effective training resources available?	Middle	Consultants Development managers
Is there evidence that the industry is adopting the method?	Middle	Consultants Senior developers
Can current development tools be used with the proposed method?	Tactical	Senior developers
Should the new method be integrated with object-orineted design?	Tactical	Senior developers

Exhibit 3). Each proposal generates questions that must be addressed by individuals from appropriate levels of the enterprise with the experience and expertise to answer them.

For example, if the CIO suggests changing database management systems, questions must be answered that cannot be answered by the CIO alone. While a junior developer would not provide much useful information about the license terms of the proposed package, he can provide information on compatibility issues. The Chief Financial Officer is not the person to ask about problems with converting primary keys, but she certainly needs to provide answers to questions of capital asset accounting.

A senior analyst makes the recommendation that the object-oriented design techniques currently in use should be replaced with component design methods. The proposed change to the architecture needs to be evaluated from different perspectives. The people at the tactical level of the enterprise have most of the experience and expertise needed to investigate this issue (see Exhibit 4).

Exhibit 5 Expanded Element Definitions

Element Name	*Purpose*	*Architecture Form*	*Weight Unit*
Requirement	Description of an application's properties and behavior expected by a user	Use Case in CASE Tool W and Narrative in Word Processor	Paragraph count
Object Design	Model of a class highlighting public methods and interfaces	Class Diagram in CASE Tool W	Method count Object weight
Relational Table	Physical structure in a relational database	Table design in DBMS X	Column count
Test Case	Definition of initial condition, input, and expected result used to verify an application	V Corp's Test Utility using Control Path and Equivalence Analysis	Edge count Class count
Test Suite	Collection of Test Cases run in sequence	V Corp's Test Utility	Scenario count
Java Class	Unit of software compiled with Java used for intranet applications	Compiler Y version 2.1	Method count Variable count

It is very difficult for a development manager to coordinate decision-making efforts involving many levels of management. But all decisions affecting the development architecture require informed, considered input. No one person, or one level of the enterprise, can have the sole responsibility to decide. It usually falls upon you, the development manager, to create the framework in which these decisions will be made. This includes:

- Defining the proposal
- Listing the questions generated by the proposal
- Suggesting (or declaring) how the decision will be made
- Soliciting the participation of required experts
- Finding alternate sources if the participants fail to provide needed information
- Announcing the consensus (or declaring the decision) once reached
- Overseeing the implementation

Mapping Architecture to Elements

The decision patterns determined by the chosen architecture must be associated with the elements you have defined for your repository. Chapter 3 discussed building element definitions to serve as the set of building blocks of a generic product. Each of these elements will have a default architecture form (see Exhibit 5).

Every well-formed team with which I have worked has established (or accepted) a set of rules and standards. The team uses these conventions as a language. The language allows the team to function at a high level of coordination. The language increases the number of important things we do not have to think about, thus giving us more time to think about the solutions we are providing to the user.

Architecture's Dark Side

I remember a conversation with a developer at a client's site. The developer was upset that I was advocating the use of standards and a well-defined framework. The developer seemed to have taken great pride in the fact that the team was working well without formal methodology, life cycle, standards, and conventions. It was as if the team "just did good work." But as I worked with the team, I noticed an incredible consistency in the way designs were factored and the way test cases were represented. Even the style and formatting of code was nearly identical in different code written by different developers. When I asked why a particular design decision was made, why an alternative was not used, or why a table was normalized as it was, the initial answer was always the same: "That is the way we do things here."

The team had created a framework and standards that were extremely well-defined (not written down but well-communicated among the team members). The team had established and accepted its norms so well that the enforcement mechanism was simply a comment from another team member.

This kind of dynamic is wonderful. It is productive, nearly error-free, and self-sustaining. And, in this case, it was unself-conscious, allowing the team's conscious talent to be directed toward building good solutions.

> Continuity does not rule out fresh approaches to fresh situations.
>
> **—Dean Rusk**

The one caveat is that while an enterprise's architecture is important to productivity and quality, it also shields the development environment from innovations in the software and business development industry. The danger is that the enterprise becomes entrenched in a technology that works for them and fails to consider the potential benefits to be gained by adopting different technologies and methods.

> **Application development is pure problem solving. Problem solving requires a set of tenets reflecting the lessons from past successes. The most constructive tenets are evolving heuristics — not immutable law.**

You have to be sure a percentage of your time and budget is allocated to the care and feeding of the framework. Most enterprises are not research and development organizations, so the intent is not to be a beta site for every new tool presented by our vendors. But a balance must be struck to ensure that opportunities are not missed and that the framework is adaptable to a changing enterprise. Chapter 7 provides some guidance on how to monitor the environment and incorporate changes that promise to improve the development effort.

The human brain craves understanding. It cannot understand without simplifying, that is, without reducing things to a common element. However, all simplifications are arbitrary and lead us to drift insensibly away from reality.

—Lecomte Du Nouy

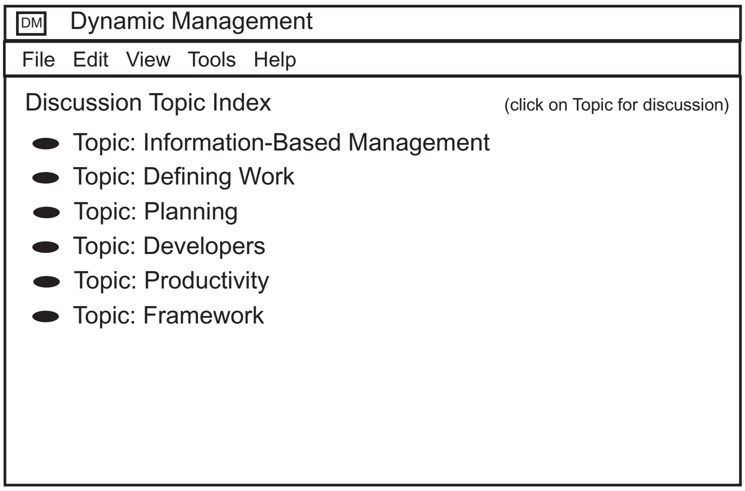
Applying Dynamic Management

Build a model of your "standard" implementation decisions. It should look a lot like Exhibit 2 but with more detail referencing the specific tools and methods you use in building software.

Compare this model with the element definitions you built after reading Chapter 3. Add a column to the element list to record the architecture forms used to build and record each of the elements (see Exhibit 5).

Watch for mismatches. If there is an element with no place in the architecture, it either means the element is not used or recorded, or there is an aspect of the current framework you have not identified.

If there is a piece of the framework that is not associated with an element, it either means that part of the framework is not used or there is some element missing from the element list.

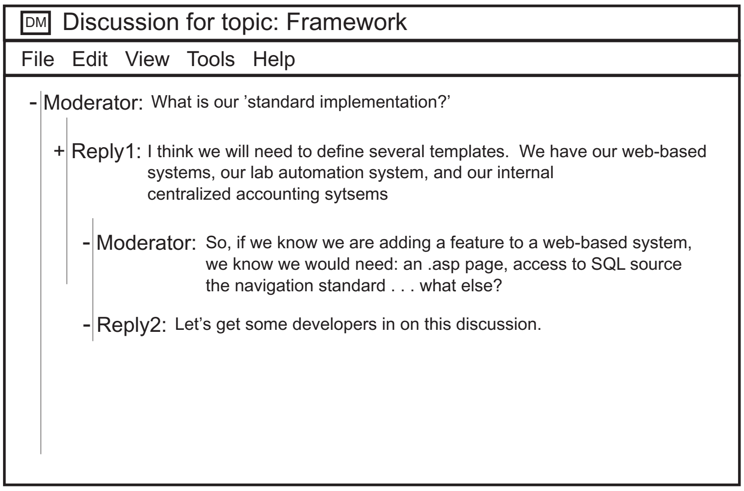

DM Dynamic Management

File Edit View Tools Help

Discussion Topic Index (click on Topic for discussion)

- Topic: Information-Based Management
- Topic: Defining Work
- Topic: Planning
- Topic: Developers
- Topic: Productivity
- Topic: Framework

Exhibit 6 Online Discussion Index



- Moderator: What is our 'standard implementation?'
  + Reply1: I think we will need to define several templates. We have our web-based systems, our lab automation system, and our internal centralized accounting sytsems
    - Moderator: So, if we know we are adding a feature to a web-based system, we know we would need: an .asp page, access to SQL source the navigation standard . . . what else?
    - Reply2: Let's get some developers in on this discussion.

Exhibit 7 Beginning Dialogue

Chapter 6 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

Shortly after the Director of Applications Development was named Chief Information Officer, he put together a Technology Task Force of senior managers and developers to review developments in information technology and development methods, and report back on the trends in the industry that DSM needed to be aware of.

The group started by surveying new technology and reading research reports on new development methods. While the work was exciting, it did not generate applicable results. It was interesting to read what the authors of new books were saying, but the members of the task force often found it difficult to translate trends into action items for DSM. The task force continued its work over the years. Every six months they issued a report.

There were successes. The work of the task force was instrumental in introducing Java to the development environment. They also got credit for aiding in the introduction of Web-based applications into the DSM framework.

Along with their search for trends in technology innovation, the task force worked with the SQA analysts to identify and define standards to be used by the developers while building work products. For example, along with the recommendation of the C++ development environment, the task force searched the literature for guidelines and best practices — including coding style guidelines and class format templates. When the task force pushed for adoption of object-oriented design, they included a proposed standard for the style and construction techniques to be used (these were based on the approach advocated by the consulting firm selected to conduct training seminars).

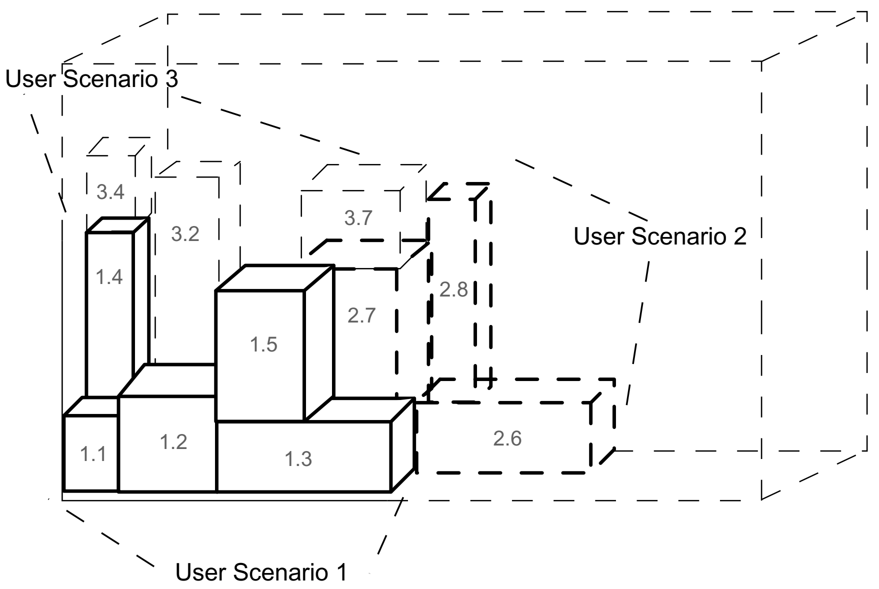


Exhibit 8 Methodology as Continuous Build (a.k.a. Chunk Model)

The task force was very active in the definition of the project repository. Its members energetically participated in the online discussion and worked with the developers on the definition of the database prototype. They tried to find ways to add their standards and framework definitions to the idea of a development repository. Several of the task force members were at the famous lunch session when the manager from Midwest drew the "chunk" model on the whiteboard (see Exhibit 8).

The blocks on the diagram became a focus for the task force for weeks after the developers' conference. The members began to talk of the blocks as mini-applications, each being implemented using a known technology set following a known set of standards.

The task force began using the vocabulary that was developing among the developers. Small clusters of these mini-applications were described by user "scenarios."

They decided there was another dimension to this model. Each of the blocks is associated with a set of default design decisions. If the block was part of the Web-based systems, the developers would use the current Web server and Web scripts. If the block included access to internal data sources, the developers would follow the audit requirements defined in DSM data administration standards.

When the developers outlined the idea of elements and element definitions, the task force proposed that each element be associated with some component of the framework (see Exhibit 9).

Some of the database designers reviewed the idea of adding the framework and standards attributes to the element table and concluded that a better

Exhibit 9 Proposed Element Definitions with Framework

Element Name	*Purpose*	*Framework*	*Standard*
Requirement	Description of an application's properties and behavior expected by a user	Use Case in T-Soft	DSM-422
Object Design	Model of a class highlighting public methods and interfaces	Class Diagram in T-Soft	DSM-31
Relational Table	Physical structure in a relational database	Table Design in SQL Server	DSM-19
Test Case	Definition of initial condition, input, and expected result used to verify an application	Boundary Analysis in Word document	DSM-485
C++ Class	Unit of software compiled with V- Studio used for our client/ server applications	Compiler Y version 2.1	DSM-60

Exhibit 10 Examples of Framework Association Definitions

Association →			*← Association*		
Element	*Meaning*	*Avg. Freq.*	*Avg. Freq.*	*Meaning*	*Element*
Requirement	recorded as	1	1	represents	Use Case
Object Design	recorded as	3	15	represents	Class Diagram
Relational Table	implemented as	1	1	implements	SQL Table
Test Case	based on	1	1	used to derive	Boundary Analysis
C++ Class	compiled by	1	n	used to compile	Compiler Y
Use Case	guided by	1	n	std. used for	DSM-422
Class Diagram	guided by	3	n	std. used for	DSM-31
SQL Table	guided by	1	n	std. used for	DSM-19
Boundary Analysis	guided by	1	n	std. used for	DSM-485
C++ Class	guided by	1	n	std. used for	DSM-60

design would be to treat the items of the framework and the standards as elements in their own right. Each item of technology (such as Boundary Analysis and Use Cases) and each standard (e.g., DSM-422 defining the qualities of Use Cases, and DSM-485 defining the characteristics of boundary analysis test cases) would be defined as elements.

Association definitions would then record the connection between the elements defined by the developers, the technology used to record/build the elements, and the standards describing desirable characteristics of the elements. A sample of the association definitions is shown in Exhibit 10.

The members of the task force found this idea intriguing. One of the managers on the task force commented, "This will make our standards an integral part of the process rather than an afterthought."

Chapter 7

Constructive Development Environment…

Or Making Work Flow

A major duty of the development manager is to ensure that the development team has an effective, integrated, smoothly functioning environment in which to work. This is, of course, a noble goal but difficult to achieve. After all, "effective," "integrated," and "smoothly functioning" are nebulous terms.

> We cannot decide whether a misfit has occurred either by looking at the form alone, or by looking at the context alone. Misfit is a condition of the ensemble as a whole.
>
> **—Christopher Alexander**

Conflict within the Environment

Most development organizations are not wanting for ideas about effective development techniques or standards or training. There are hundreds of consulting firms with many useful techniques for improving the development environment. Thousands of books and articles provide advice to managers on which tools to buy, what methods to adopt, and which education programs are best. Most of the counsel is reasonable and well-meaning.

During most of my early consulting, I was troubled and frustrated by a ubiquitous condition. Despite the best efforts of talented managers and developers — despite the best advice of experienced consultants and academicians — organizations could not get their development environments

Exhibit 1 Conflicting Patterns

"We have purchased this great tool."	→	←	"We'll try to schedule training next quarter."
"You need to do a thorough specification before building the design."	→	←	"Why aren't you programming!?"
"Schedule a review of your code."	→	←	"Ignore the findings of last week's review. We have to deliver the product."
"We are going to use these new standards."	→	←	"The tools we are using do not support that standard."
"We measure defect rates."	→	←	"Joe gets the promotion because he gets his work done fast."

to work smoothly. Standards never seemed to deliver on their promise of consistent results. Methods always resulted in too little work, too much work, or the wrong work for the targeted activity. Training never succeeded in enhancing the developer's skill set to the degree expected.

Most managers become rather cynical about new methods and standards after having put in so much effort for so little return. Most come to the conclusion that the consultants and the academicians do not know what it is like in the trenches and blame the communication gap between theory and practice.

When the map and the terrain disagree, trust the terrain.

—Swiss Army Proverb

There is another explanation for the phenomenon. The conflict arises in the way components in the development environment "fit." Taken individually, each component of the development environment sounds reasonable, but in combination they often result in unintended conflict. You can identify the conflicts just by listening to conversations in the hall (see Exhibit 1).

Just as elements of a software product must fit together, the environment in which the elements are built must fit together.

Conflicts in the development environment inevitably diminish our ability to build effective software solutions. Removing dissonance among components must necessarily enhance our ability to do good work.

For example:

- The tools used by the developers are consistent with the standards adopted by the department.

- The measurements used to assess quality are employed in product reviews.
- A product produced by the development team is defined in the project's methodology.

Conflicts between elements in the development environment introduce obstacles to the developer's progress. For example, introducing a technique for exploiting common functions and reducing the volume of new code should encourage developers to look for ways of increasing reuse. But a conflict is created if the department's productivity measurements are based on size and the developers feel that their evaluations are partially based on creating measurable volume. A developer who succeeds in increasing reuse comes out poorly when compared to a colleague adding lots of visible stuff to the repository. The initial conclusion might be that the reuse technique did not work, but the real cause of failure is the dissonance between the technique and the measurement.

> I should like to recommend that we should always expect to see the process of achieving good fit between two entities as a negative process of neutralizing the incongruities, or irritants, or forces, which cause misfit.
>
> **—Christopher Alexander**

Conflicts often occur between development methodologies and standards. An organization builds or adopts a methodology defining a task and associated work product. The standards describing the quality characteristics of the work product refers to a second work product (i.e., specifies consistency requirements between the work products). If the methodology specifies that the second work product is to be built at a different point in time or by a different team, the developer must complete the first product in isolation — making assumptions about the related product. When the standard finally is applied, the first product may be entrenched and developers reluctant to entertain the idea of making changes suggested by the standard. The conflict here is not attributable to the methodology or the standard. Both may seem reasonable and constructive when considered separately; but in combination, they have a destructive effect on the development effort.

Seven Components of a Development Environment

The development manager must review the development environment in an attempt to identify conflicts that hinder progress. There are seven components to consider: standards, tools/techniques, methodologies, measurements, rewards, reviews, and training/education.

A *standard* is a statement of the observable attributes of an acceptable product. A standard specifies the observable characteristics of good work. Most standards have a lot of verbiage describing the standard's history and its importance. But the portion of a standard important to developers is the description that helps distinguish good products and actions from not-so-good products and actions.

Tools and techniques are software and procedures used to aid and direct the development work. I can define tools and techniques separately, a tool being the instrument with which work is performed, and technique being a set of principles or guidelines used to perform work. I combine them because, in practice, they are often inseparable. Tools like computer-aided software engineering tools or code complexity analyzers or language compliers are usually based on a defined technique. The CASE tool may be designed to support with Rational Unified Process or Dynamic System Development Method. A code analysis tools will be based on techniques developed by McCabe or Halstead. Language-based development environments usually support some techniques like object-oriented or component design.

A *methodology* is a set of task definitions. Each task definition identifies necessary pre-conditions and expected results. Most methodologies have a lot of detail about the sequencing of tasks and content of input and output documents. I classify most of this detail as "technique" and "standards," as defined in the previous paragraphs.

A *measurement* is an observable, objective scale (quantitative or qualitative) used to assess quality, volume, or duration. Measurements can be very formal, like function point and edge counts; or they can be very informal, like promptness to meetings and contributions to the coffee fund. Any measure that the developers believe is being used to assess work and performance is relevant to us.

Rewards include any means of recognizing and encouraging desirable behavior or results. Bonuses, promotions, formal "thank you" communiqués, job assignments, and office space can all serve as rewards.

A *review* is any means of assuring the quality of a product and the adherence to standard or plan. Product inspections, peer reviews, and code walk-throughs are common forms of reviews. Extreme programming is introducing the idea of continuous review.

All people involved in the development effort must have access to and encouragement for training and education. My working definition of *training/education* is a means of acquiring working knowledge or useful information.

> I have been involved in many debates over the difference between training and education. Frankly, the distinction is arbitrary for the purposes of managing a development effort.

Sources of Conflict

Conflict or dissonance arises in the development environment when a developer, working in good faith toward a legitimate goal, finds himself in a situation

Exhibit 2 Inter-component Influences

	Influences						
Component	*Rewards*	*Reviews*	*Methodology*	*Standards*	*Measures*	*Education*	*Tool/ Techniques*
Rewards	—	Strong	Strong				
Reviews	Strong	—	Strong	Strong			
Methodology	Strong	Strong	—	Strong			
Standards		Strong	Strong	—	Strong		
Measures	Strong		Strong		—		
Education		Strong	Strong			—	Strong
Tool/ techniques				Strong	Strong		—

in which his work is supported by, and consistent with, one component in the environment, but in conflict with, or hindered by, another component.

For example, the development team is using a code complexity measurement. What conflicts can exist?

- The coding standards may require constructs that are measured as overly complex.
- The development team may be using a tool that generates components with interfaces that are measured as overly complex.
- The developer may be rewarded for delivering the code by the end of the week although the code is overly complex.
- The measure may not be included as a significant issue in the product review.
- The developers may not have the training to know how to work constructively with the measure.

Any identified conflicts suggest that some action is needed to remove the dissonance. To reduce the dissonance between the review and the measurement, the manager could require an assessment of the code complexity in the review report. Providing education on the measure's meaning and use would also improve the development environment.

Instances of each of the seven component types can be in conflict with any other component in the environment. Imagine a 7 × 7 matrix where each of the 49 cells represents a potential conflict. That is more than I can handle. You can focus the assessment effort by identifying relationships between components that you and your developers find to be most significant. My experience is summarized in Exhibit 2. The same set of relationships are graphically depicted in Exhibit 3.

Exhibit 2 should be read as follows: if I have a problem with a component listed on the top row, I should look to strengthen support from strong component types along the left side. Issues related to rewards might imply a problem with the reviews, methodology, and measures. I can strengthen the

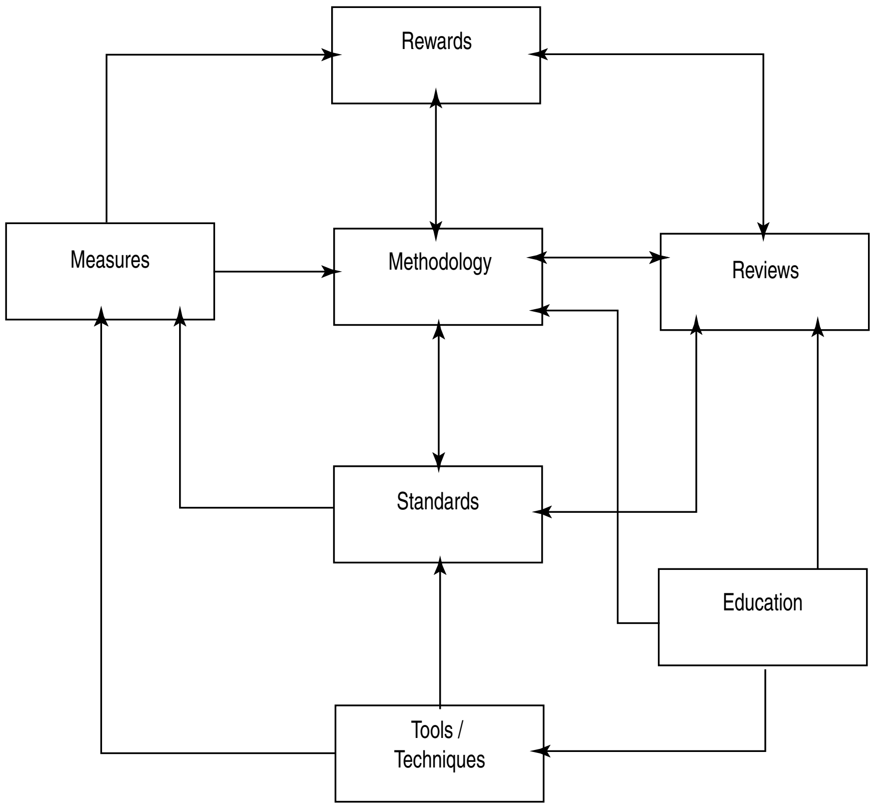


Exhibit 3 Components of the Development Environment

reward system by employing reviews to identify the good work that should be rewarded. I should use the methodology to identify the activities to reward. I should be sure that the things I am rewarding are consistent with the measurements in place.

If it seems that the reviews are not as effective as they could be, I should try to strengthen the reward structure. If I have few rewards in place for reviews, reviews deteriorate. Reviews might be enhanced by ensuring that the methodology defines logical points for reviews to occur and well-defined products to review. Problems in the reviews might trace back to ambiguous standards against which products should be evaluated. Enhancing education could help resolve issues of poor or ineffective reviews.

The use of methodologies can be strengthened (or undermined) by many factors. If the methodology I am using is becoming less effective, it might mean that the rewards for working within a methodology are weak. Methodologies can deteriorate if there is no method of checking the work done and gaining consensus that good progress is being made. A lack of standards can contribute to inconsistent results that can undermine the use of a methodology. Similarly, the lack of objective measures can make it difficult to track progress in a demonstrable manner. And, as with reviews, if the developers and managers do not have the skills and knowledge necessary to work with the

Exhibit 4 Component Inventory

Name	*Type*	*Description*
ABC CASE tool	Tool	Modeling tool used for building specifications and designs
Java Development Kit	Tool	Develop language
OOA workshop[a]	Education	Workshop conducted by consulting firm for new developers
Defect rate	Measurement	Track number of defects reported by users
Peer programming	Review	Part of the extreme programming pilot
Java style guide	Standard	Conventions established by developers

OOA = object-oriented analysis

methodology, the methodology will not contribute to an effective development effort.

The use of measurements makes more sense when they are supported by objective standards and tools that assist in the gathering and analysis of the data.

Standards can be a positive factor if they are supported by a complementary methodology, reviews that help enforce the standards, and tools that aid the developers in building product consistent with the standards.

One biggest point of dissonance in the software development industry is the mismatch between tools and training. Often, an organization will purchase potentially effective tools only to render them nearly useless by not supporting them with education.

Assessing the Environment

These seven components form a framework for assessing your environment. The assessment process entails analyzing each element in the development environment and ensuring no other element is in conflict.

Start by building an inventory of the components in your development environment (see Exhibit 4). The inventory is then analyzed by reviewing each component, trying to identify potential conflicts. The algorithm is: the list of identified conflicts (a dissonance list) represents opportunities for improvement. Not all conflicts are fixable or even a real problem. You want to choose your battles and identify changes to the development environment that have a high probability of success with little risk and not much money.

> The development environment is improved by small and frequent victories.

Perhaps in your assessment of your environment you find that the use of the CASE tool is weak and you believe the lack of training is a factor (see Exhibit 5). You also find that programmers on a pilot of extreme programming techniques are not supported by a methodology. You think that might be a problem.

Exhibit 5 Dissonance List

Issue	*Possible Action*	*Expense*	*Difficulty*	*Anticipated Result*
ABC CASE tool has no training support	Contract with vendor to deliver in-service training	Expensive	Easy	Little real effect
	Arrange "best practice" seminars by developers	Not expensive	Requires management support	Potential for positive response
Peer programming (as a review) is not supported by methodology	Impose a strict waterfall methodology on the extreme programming team	Not expensive	Not hard	Open rebellion

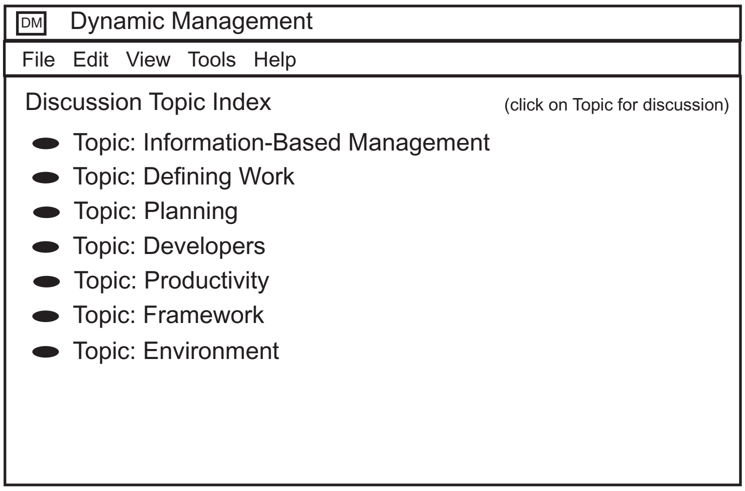
For each of the issues you find, you need to identify potential remedies. Identify as many possible actions as you can for each issue. For each of the possible actions, you can project the expense and degree of difficulty (you are trying to assess your chances for success). Before taking any action, you want to think about the possibility of success and convince yourself that the anticipated result is truly better than the current status of things.

For example, you consider that one possible remedy for lack of educational support of the CASE tools is to hire a vendor to conduct in-service training sessions. But if your experiences with the vendor's training is less than stellar, perhaps the anticipated result is not worth the price. Another option is to set up time for developers to share their "best practices." The difficulty would be ensuring support for the developers and management acceptance of the time in the self-training sessions as legitimate and valuable work.

The issue with the lack of methodology might be a different story. If the imposition of a strict phased-based methodology resulted in open rebellion among the developers, the value of the move is in question. Perhaps the lack of a supporting methodology is not a problem. If you can define measurable results from the team, it might mean there is no real conflict between the extreme programming and methodology and the environment should be left alone.

Ongoing Assessment

The development of software solutions is an ongoing, never-ending, iterative, and perpetual activity. So too is the assessment and refinement of the development environment. As new components are introduced, the inventory changes and the balance within the environment is altered. After the initial



Discussion Topic Index (click on Topic for discussion)

- Topic: Information-Based Management
- Topic: Defining Work
- Topic: Planning
- Topic: Developers
- Topic: Productivity
- Topic: Framework
- Topic: Environment

Exhibit 6 Online Discussion Index

inventory and evaluation, you need to regularly monitor the set of components that make up your development environment and gauge its overall balance.

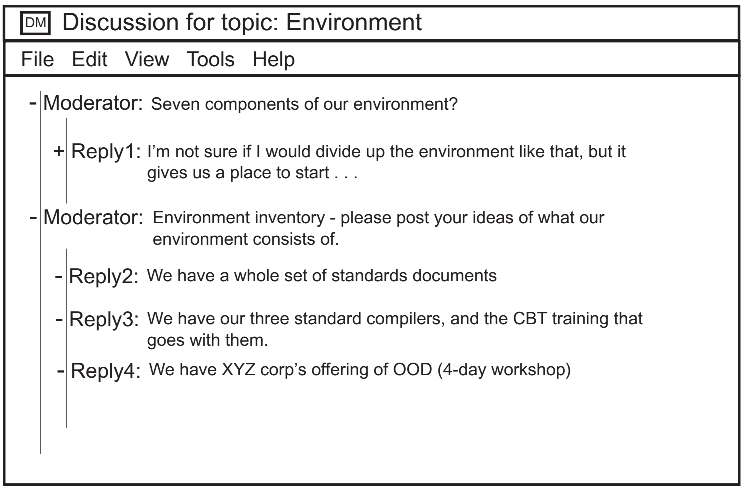
Use the idea of a conflict-free environment when considering any change. A vendor might introduce a new development tool. Assess the dissonance of the environment if a new tool were to be added. Any dissonance means disruption, cost, and potential error. Each point of dissonance means time, money, and energy must be applied to deal with it. If each of the points of dissonance can be addressed at a reasonable cost, introducing the new tool is a constructive change. If the dissonance cannot be removed or the cost of removing it is too high, introducing the new tool is a mistake — regardless of how wonderful the tool is on its own merits.

Applying Dynamic Management

Discuss the development environment with your developers. Get their thoughts on the seven-component model. Make changes to the model to bring it in line with the mental model of your environment and the view your developers hold (see Exhibits 6 and 7).

Build an inventory of components being used to develop your software (i.e., do not invest in new computer-based information system — just create a list). Run through the algorithm looking for potential conflicts.

```
For each component in the inventory (call this component A)
   For each of the other components (call this component B)
      Think about how component A supports or conflicts with
         component B
      Record any conflicts in the inventory's dissonance list
   End For
End For
```



Discussion for topic: Environment

- Moderator: Seven components of our environment?
  + Reply1: I'm not sure if I would divide up the environment like that, but it gives us a place to start . . .
- Moderator: Environment inventory - please post your ideas of what our environment consists of.
  - Reply2: We have a whole set of standards documents
  - Reply3: We have our three standard compilers, and the CBT training that goes with them.
  - Reply4: We have XYZ corp's offering of OOD (4-day workshop)

Exhibit 7 Beginning Dialogue

Verify that the conflicts are perceived by your developers and judge the degree to which the conflicts hinder your development team's effectiveness. This step is best done in information strategy sessions with small groups of managers and developers.

Prioritize the list of conflicts, ranking issues that can be reduced with simple and inexpensive action. Identify the results you expect from taking the action. This step is best done by you — alone — after having the advantage of good counsel from the previous step.

Choose the most promising actions (i.e., low cost, high probability of success) and implement them.

Chapter 7 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

The experiment in incremental development seemed to be working. There was general consensus among managers and developers on the West Coast and at Midwest that focusing on the repository was a viable alternative to focusing on the process. People were recording application elements in the repository and they felt good about the accuracy of the information. Recording elements of the development framework in the repository was also working. DSM products were more consistent and innovative because developers were able to work from a template of solutions. Even the idea of defining the developer's goal as enhancing the state of the repository and maintaining internal consistent had become comfortable.

However, another concern developed. When work was performed as a sequence of processes, each process was governed by a set of standards and techniques. The task force had developed these standards and defined them

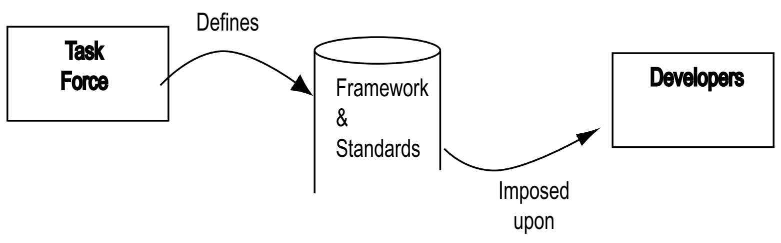


Exhibit 8 Task Force Defines Framework and Standards

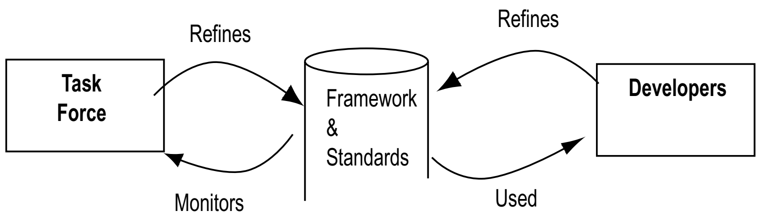


Exhibit 9 Shared Maintenance of Framework and Standards

as a fixed part of the process (see Exhibit 8). Developers were not expected (nor did they have any opportunity) to question the standards or contribute to their refinement. The task force would, from time to time, review the standards and make updates, but that refinement was based on task force's findings, and not on the experience of the developers.

Once the items of the framework and the standards were represented as elements in the repository, they became just another part of the knowledge base. The goal is to create consistency. When faced with a conflict between a design and related code, developers would refine the design or the code or both — choosing the best solution. Now, when faced with a conflict between a design and the design standards, developers might find the best solution is to change the design or the design standards or both. The objective is to create an internally consistent set of information about the product. When the developers identified a change to a standard they believed would be constructive, they wanted to be able to change it (see Exhibit 9). Members of the task force were concerned about their role in the development process and argued that the developers would tend to soften and diminish the standards. The developers argued that their experience in applying the standards yielded positive changes to the standards that deserved to be considered.

For the past year, conflicts between the developers and the members of the task force were few and were handled one at a time with minimal disruption. But finally, a group of developers from the Midwest group proposed a major revision to the code documentation standards and insisted upon documenting their code accordingly. The task force refused to consider

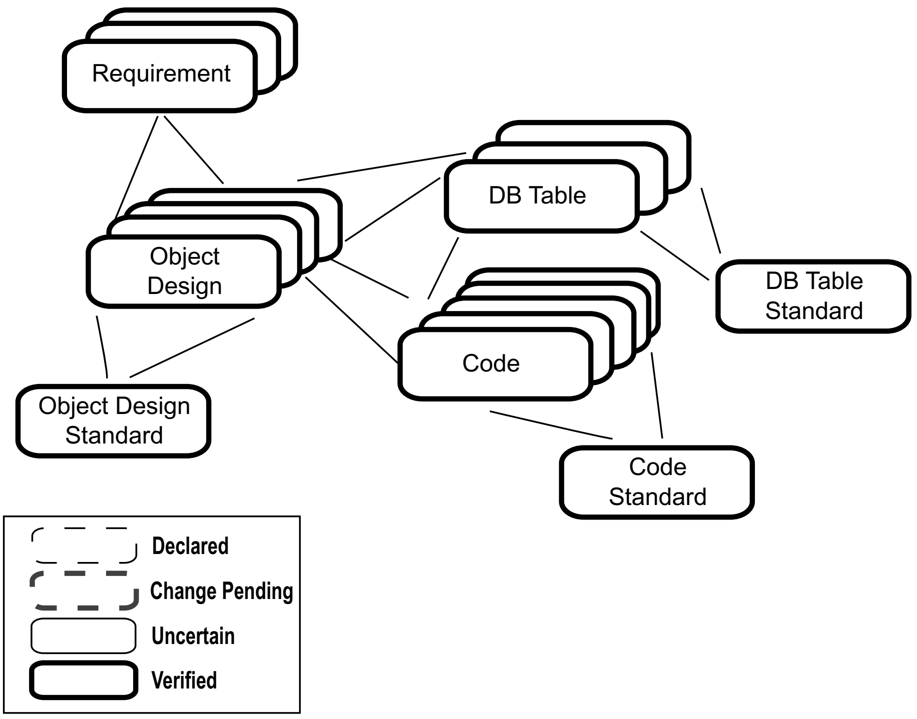


Exhibit 10 Current State

the proposal on the grounds that it would result in a different set of documentation than all DSM's older software had used.

The CIO stepped in when he heard the conflict was growing out of proportion. His response was rather unusual. He resisted the temptation to call a meeting with all parties present to "duke it out." He had seen meetings like that end with each side being obliged to defend its initial position even after compelling arguments were presented. The CIO went to the office of each of the task force members and called each developer on the Midwest team. During these private conversations, the CIO asked each individual to explain his proposal and to explain the other side's proposal. He discovered that everyone thought the changes were good and constructive. The only issue from the task force was the fact that existing code and new code would be documented differently.

After the discussions, the CIO met with the task force as a group and summarized the individual conversations and suggested that the task force modify the current code standards, incorporating the suggestions from the Midwest developers. He told the task force members that change is inevitable and that they would have to accommodate many generations of techniques, standards, platforms, and product. The CIO then opened up discussion about how to handle the issue of elements being built in different time frames under different standards. This was a safe question because two task force members

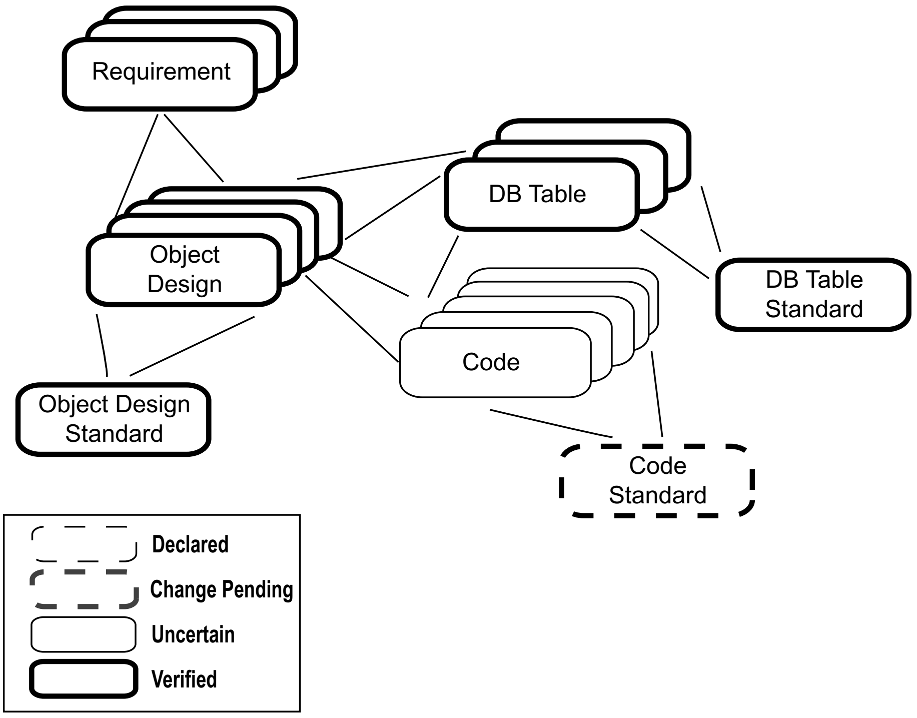


Exhibit 11 Start after New Standard Is Approved

had outlined their ideas during the one-on-one conversations. For example, one task force member had suggested that nothing be done to the existing code elements. The other member suggested the existing code could be reviewed against the new standard over time (see Exhibit 10).

Approving the new standard changes the state of all associated code elements to "Uncertain" (see Exhibit 11). The elements may not need any change but, technically, it is possible that an inconsistency exists between the old code and the new standard that should be reviewed.

No separate development plan would be created. The code would be reviewed only when it was part of another development task. That is, for each future enhancement, developers will be required to review the code in its strand against the new standard. After the code is reviewed and possibly changed, the state of the code element would changed back to "Verified." Everyone agreed this was a good plan.

The CIO then showed the members of the task force a memo he was sending out that day, stating that suggested changes to DSM techniques and standards were welcome and encouraged from all managers and developers. The recommendations should be directed to the task force. Only the task force had the authority to approve such changes. Before he left the meeting, the CIO suggested that task force members should do what each of them felt was appropriate and approve the changes to the coding standards.

Chapter 8

Managing Managers…

Or I'm OK, but the Rest of Them?

There are many managerial roles involved in building system solutions. Each enterprise has its own structure, reporting and authority lines, and policies governing the delegation of authority. A significant factor in successfully delivering effective solutions is your ability to manage the managers with whom you interact. The standard pronouncement in management texts is that coordination among managers is a matter of successful communication. But communication is more than talking; it is a process of defining, sharing, and adding meaning to information.

> Communication of all kinds is like painting — a compromise with impossibilities.
>
> **—Samuel Butler II**

Dealing with Expectations

A manager is anyone in an enterprise with the authority to direct and allocate resources. Every manager has a set of objectives (both personal and professional). The resources allocated to the manager are used to help achieve his objectives. Resources and the objectives are delegated to subordinates, who may in turn delegate subsets to their subordinates, thus forming the typical organizational hierarchy. Along with that delegation comes the need to coordinate the superset of objectives and resources with the subsets being managed by subordinates/colleagues. This is depicted in the relationships A to a1 and A to a2 in Exhibit 1.

Middle managers have to collaborate with others within their division on their respective pieces of the overall organization objective. Redundancy,

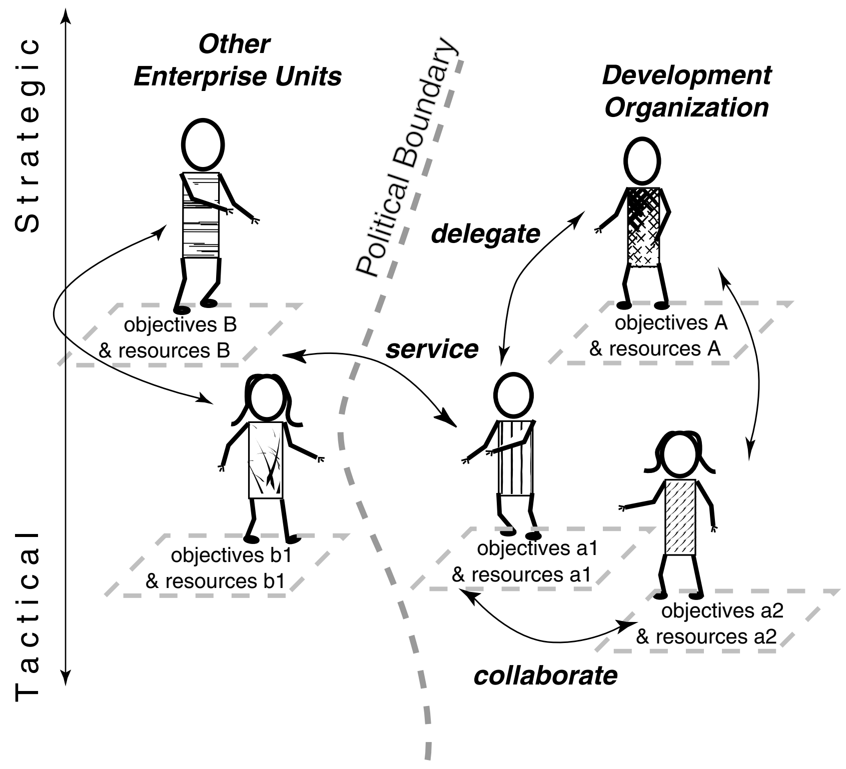

Strategic
Tactical
Other Enterprise Units
Development Organization
Political Boundary
delegate
service
collaborate
objectives B & resources B
objectives A & resources A
objectives b1 & resources b1
objectives a1 & resources a1
objectives a2 & resources a2

Exhibit 1 Managerial Relationships

inconsistency, and mismatched resource allocation can reduce the chances of successfully achieving the organization's objectives. This relationship is shown as the interaction between a1 and a2 in Exhibit 1.

Collaboration across the organizational boundaries established by the delegation is also required (i.e., any interaction A to B in Exhibit 1).

The vectors in Exhibit 1 depict the three communication channels significant to every manager:

- Delegate
- Collaborate
- Service

Every person in an enterprise has expectations concerning the relationship between themselves and the people with whom they work. These relationships take the form of information sharing.

The easiest way to think about these relationships is to define them as a series of messages (request and response; see Exhibit 2). A CIO directs one of his managers to study the feasibility of a proposed enhancement (this is a request on the delegate channel). The manager talks with a colleague about related elements of the proposal (request on the collaborate channel). The colleague provides some historical information (response on the collaborate

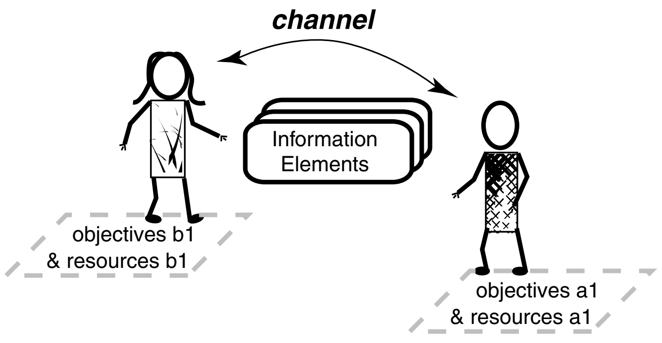


Exhibit 2 Communication Channels as Elements of Shared Information

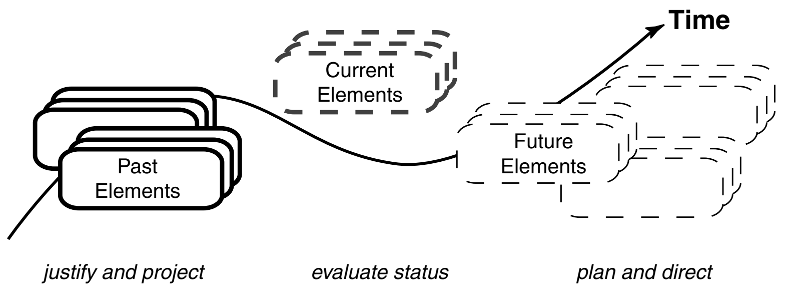


Exhibit 3 Shared Elements (Past, Current, and Future)

channel). The manager makes an appointment to talk with a user in a different department. During the meeting, the manager asks for the user's thoughts on the benefits of the proposal (request on the service channel). The user provides her opinion and analysis (response on the service channel). The manager summarizes the information gained from the interactions, formulates an opinion, and reports his findings to the CIO (response on the delegate channel).

While this scenario is reasonably accurate and convenient to define, it is an inadequate model for helping us plan and facilitate the development effort. Instead of thinking of these channels as conveyor belts along which discrete packages of information flow, think of these channels as shared information that sits between the players (see Exhibit 3). The information exists over time. Both parties involved in the channel can think about the information at any time. Both parties can add to and modify the information at any time.

This shared information is not unlike the element defined in Chapter 2. These elements are more difficult to label and define. It is easy to define the meaning and attributes of a relational database table. We are less precise when we are trying to define the meaning and attributes of a "corporate objective." Typically, these elements are not stored in a database or tracked under configuration management, but the notion is very useful in coordinating managerial activity — just as it is in coordinating teams of developers.

Exhibit 4 Delegate Questions

Roles	*Element/Questions*	*Roles*
Vice-presidents Directors	**Current** Are resources being used well? Are resource changes required? Are objectives reasonable?	Development manager Development supervisor
	Past What lessons should we remember from past efforts? Is there information in past efforts that will help project time and resource requirements?	
	Future What resources will be needed? Are there better objectives?	

The shared information resonates and is refined by all individuals involved. As time moves along, the information shared between managers is drawn upon to assess current status, justify and rationalize the results of past actions, and plan future efforts. It is through these shared elements that expectations are created. Your ability to manage the expectations of other managers depends on your ability to manage these elements (i.e., create and modify this shared information).

Most of the elements in the channels can be constructively envisioned as the answers to interesting questions. Two development managers will share the answers to the question, "What is the current status of the product pricing enhancement?" A development manager and a user will share the answers to the question, "What did we try in the past to address the pricing issue?" A manager and the CIO will share answers to the question, "Should we upgrade to the next version of the DBMS?"

The Delegate Channel

It is through the delegate channel that you support higher levels of management (and maintain their support). The characteristics of this channel are determined by the more senior role. I have seen relationships being very formal and regulated, while others have been more casual. But always (and understandably), the tenure of the discourse has been similar to the style of the upper-level manager. This channel was created by the delegation of resources, so the usual subject of the elements is the effective use of these resources toward the organization's objectives. A sub-theme of the dialogue is the objectives themselves (see Exhibit 4).

The information shared in the elements is never complete, nor will it be 100 percent accurate. This is not usually a problem. The elements are perpetual and constantly in flux. It is in your best interest to monitor the state of these elements (just as you monitor the state of elements in the development repository). If these elements are not being addressed, you will need to take steps to raise the issues by suggesting answers and inviting comment and criticism. Dialogue is the important thing. The objective is considered discussion and consensus, not having the "right" answer every time you speak.

A little inaccuracy sometimes saves tons of explanation.

—H.H. Munro

The Collaborate Channel

The collaborate channel is used to synchronize efforts with other managers within the development organization. This channel involves sharing information and supporting each other's allocated objectives. I find that this channel is characterized by adjectives such as "mutual," "adaptive," and "implicit."

Given that the objectives delegated to each manager originate from the same source, it is likely that they are compatible and both managers recognize the mutual benefit to be gained from coordinating their efforts. One would hope that the objectives are not mutually exclusive, causing counterproductive competition for resources and support of the other managers. In any case, the dialogue should address the relationship between the collaborating managers (see Exhibit 5). The conversation can be malleable and adaptive because there is typically a degree of freedom in the way responsibilities are shared. Most of the managers with whom I have worked have felt they have autonomy to share resources, trade assignments, or even alter directives if everyone agrees and the net effect is the same as the original objective. Managers communicating across a collaborate channel are usually the most collegial of the three channel types. While delegate channels can be formal and service channels follow a fixed protocol, the collaborate channel is based on rules that are implicit and often altered to mold to the personalities of those involved.

The Service Channel

User involvement and feedback occur along the service channel. I characterize this channel as transactional. It tends to be formal but cordial. The way the development organization interacts with other enterprise units is usually negotiated. A development manager may talk with the marketing manager in formal meetings scheduled ahead of time. The development management may meet with the human resources manager at the local coffee shop before work. The subject matter of the elements in this channel tends to focus on the services the development organization is providing. Most of the discourse concerns the current status of the applications, peppered with suggestions and questions about possible future services (see Exhibit 6).

Exhibit 5 Collaborate Questions

Roles	*Element/Questions*	*Roles*
Development manager Development supervisor	**Current** What are your developers doing? Does it affect me? Are we performing as well as we have in the past? Are our current status of our tasks in line with our objectives? **Past** Have we done similar tasks before? Are the assumptions we made in the past still valid? Is the historical data we are using based on the same assumptions we are using today? **Future** How can we organize ourselves to meet objectives? Should we alter the way we do things? Are the resources we have adequate to the anticipated tasks?	Development manager Development supervisor

Exhibit 6 Service Questions

Roles	*Element/questions*	*Roles*
User director User manager User supervisor	**Current** Is the service constructive? What problems are we encountering? Do we see any opportunities for improvement? **Past** Have services you have been using been positive? Is there evidence that benefits we expected are being realized? **Future** Will proposed changes work with current services? How will we prioritize the proposed changes?	Development manager Development supervisor

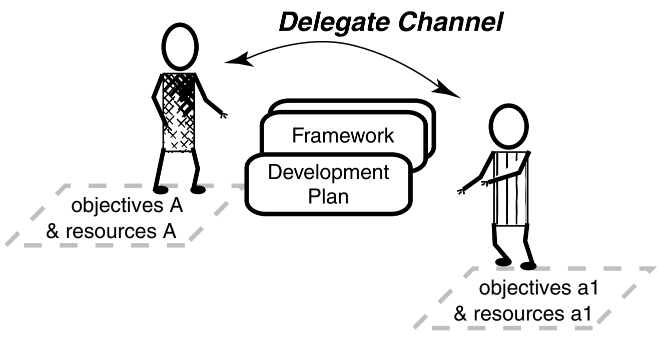


Exhibit 7 Delegate Channel

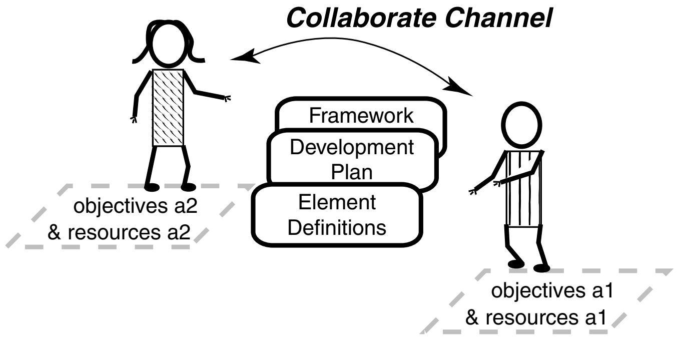


Exhibit 8 Collaborate Channel

The best way to ensure support for ongoing development efforts is to take great care in supporting the communication channels (shared information) between you and the other managers in the enterprise. Most of the failures I have witnessed have been directly associated with failure to recognize these shared information elements as critical assets that deteriorate if they are not tended.

Selling Enhanced Value

Understanding the communication channels that connect you to other managers is important. Using those communication channels to encourage creative work is critical. Managerial communication is always enhanced when information is freely available and accurate.

The information discussed in previous chapters forms a valuable focal point for all three communication channels (see Exhibits 7, 8, and 9).

The resources that are being managed through the delegate channel include the developers and the equipment and tools. The information is the Framework (refer to Chapter 6) and the Development Plan (refer to Chapter 3). With both

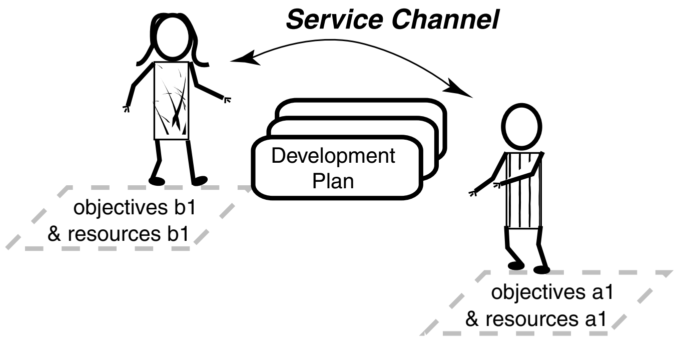


Exhibit 9 Service Channel

parties focusing on the same, accurate information, they stand a better chance of making effective, constructive decisions about the questions in the channel.

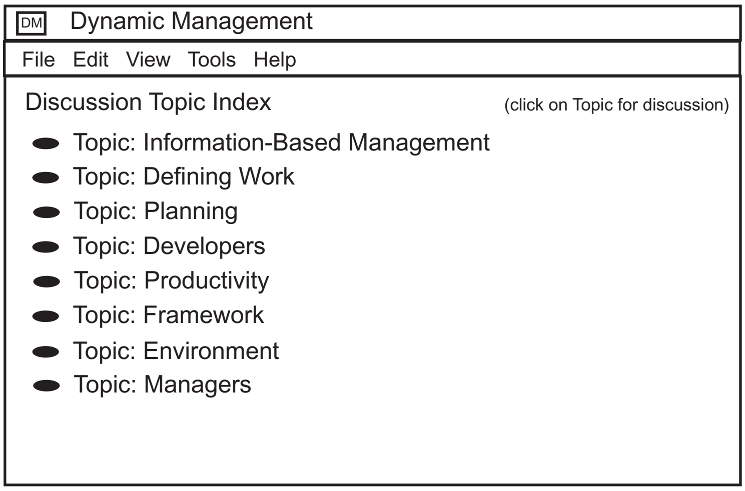
Development managers address issues of the development efforts, the framework on which applications are built, and the environment supporting the development. The Development Plan and Framework as discussed in Chapters 6 and 3, respectively, provide a common knowledge base for tactical decisions that must be made. Issues relating to the structure of the repository were discussed in Chapters 2 and 4. Using the development repository can provide a common basis for evaluating status and planning ongoing development.

Managers in user departments of an enterprise are concerned about the service they receive from the development organization and how those services might affect their success (both positively and negatively). The discourse occurring on the service channel centers on the value the software provides and plans for integrating proposed changes into operations. The Development Plan has information concerning the tasks in progress and projections of the development schedule. This information should be used to focus and direct the service channel discussions.

Applying Dynamic Management

Draw an organization chart for your organization. Classify each of the communication channels with your managerial colleagues as "Delegate" or "Collaborate" or "Service" (see Exhibits 10 and 11). For each of the channels, list the questions you are currently investigating with that person. Record questions you recall addressing in the past. If you feel there are questions that will come up in the foreseeable future, record these and begin to list the sources of information you might tap to address them.

You may find it useful to start three separate discussions — each aimed at evaluating effective ways of communicating along the three channels: delegate, collaborate, and service.

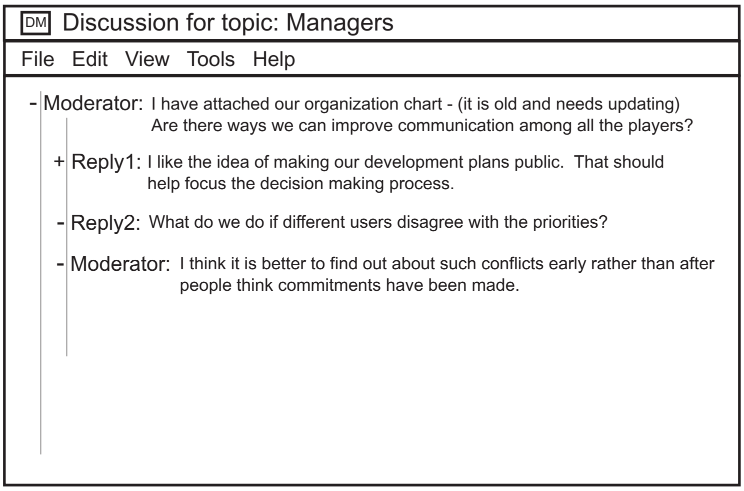


Dynamic Management

Discussion Topic Index (click on Topic for discussion)

- Topic: Information-Based Management
- Topic: Defining Work
- Topic: Planning
- Topic: Developers
- Topic: Productivity
- Topic: Framework
- Topic: Environment
- Topic: Managers

Exhibit 10 Online Discussion Index



Discussion for topic: Managers

- Moderator: I have attached our organization chart - (it is old and needs updating) Are there ways we can improve communication among all the players?
  + Reply1: I like the idea of making our development plans public. That should help focus the decision making process.
  - Reply2: What do we do if different users disagree with the priorities?
  - Moderator: I think it is better to find out about such conflicts early rather than after people think commitments have been made.

Exhibit 11 Beginning Dialogue

Universally fundamental laws of literary communication:

1. One must have something to communicate.
2. One must have someone to whom to communicate it.
3. One must really communicate it, not merely express it for oneself alone.

Otherwise, it would be more to the point to remain silent.

—Friedrich Von Schlegel

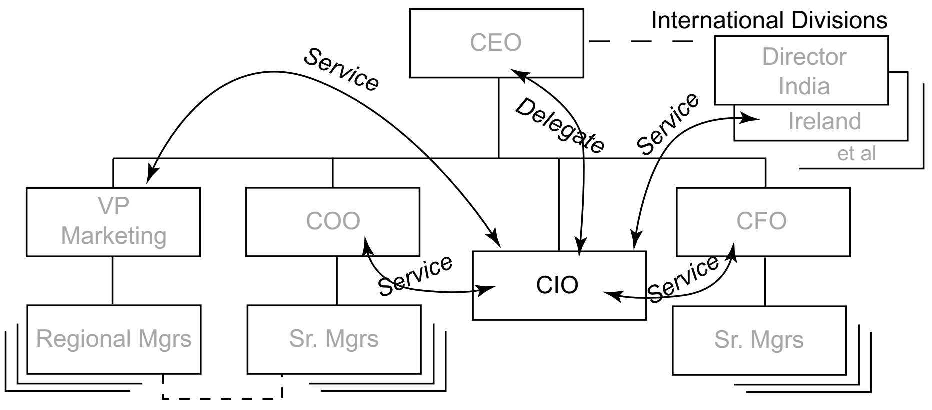


Exhibit 12 Communication across DSM

Chapter 8 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

When the Director of Applications Development was named the CIO, he took some time to think about his role within DSM (see Exhibit 12). His personal goal was to identify the type of information each of his constituents needed. The CIO reasoned that if he could provide the information that each colleague expected, they had a good chance of maintaining good communication, effective decision making, and more reasonable prioritization of goals. If he provided the wrong information, he was sure his effectiveness as an officer of the company would be severely compromised.

The information relationships with other officers of the company would be one of "service." That is, the office of the CIO would be judged by how well it contributed to the success of the other divisions of DSM. So, the CIO was determined to keep the other divisions of DSM informed of the status of their particular requests. He also knew that he would always have more requests for service than he had resources to fulfill them. The information he shared with other executives would have to include criteria and strategic goals he used to prioritize development requests. It would not be enough to say to the VP of Marketing that the request for an update of field support applications would not be addressed for six months. The information would also have to include what applications were given precedence and why. The decision-making process would have to be coordinated and flexible, and would need the support of his boss, the CEO. The information shared with the CEO would have to be tailored to help the CEO direct the discussion of corporate priorities.

The CIO determined that communication within the development group would be mostly collaborative (see Exhibit 13). While there were formal administrative relationships, traditionally, managers have tried to function as a member of the development team. The CIO wondered if that was really the best arrangement, but every manager had come up through the ranks of the

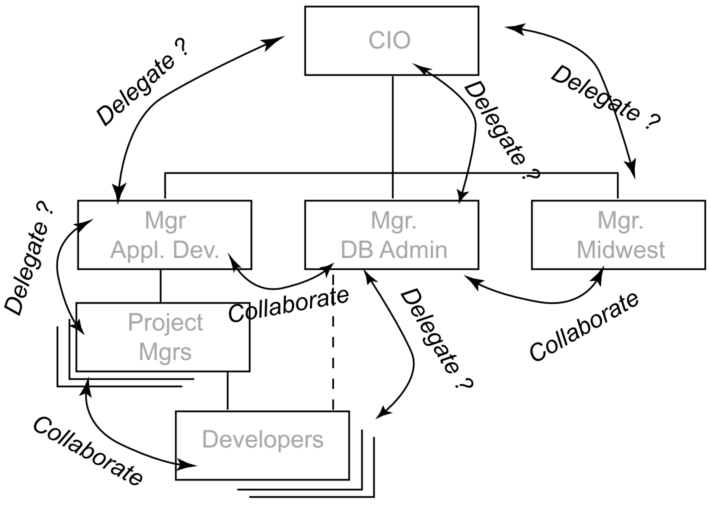


Exhibit 13 Communication within the Development Group

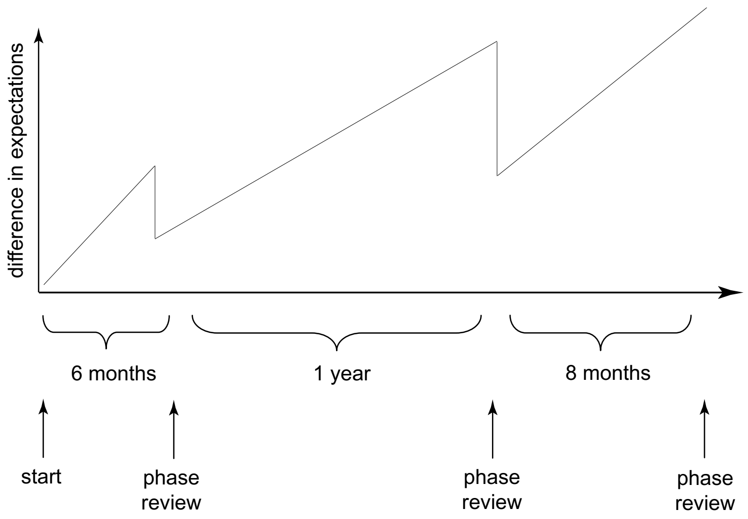


Exhibit 14 Growing Difference in Expectations

development staff and enjoyed staying involved with the technology and the development work. For these communication channels, the CIO determined to make the development plans and all elements of the environment open to debate and refinement. Decisions about standards, methods, practices, tools, and training needed the perspective of developers. Of course, decisions had to be made and it was clear that the managers would have that responsibility, but all decisions had to be informed by the lessons learned by the developers and project leaders.

Communication between the development group and the rest of DSM has always been difficult (see Exhibit 14). It seemed that the Director of Applications

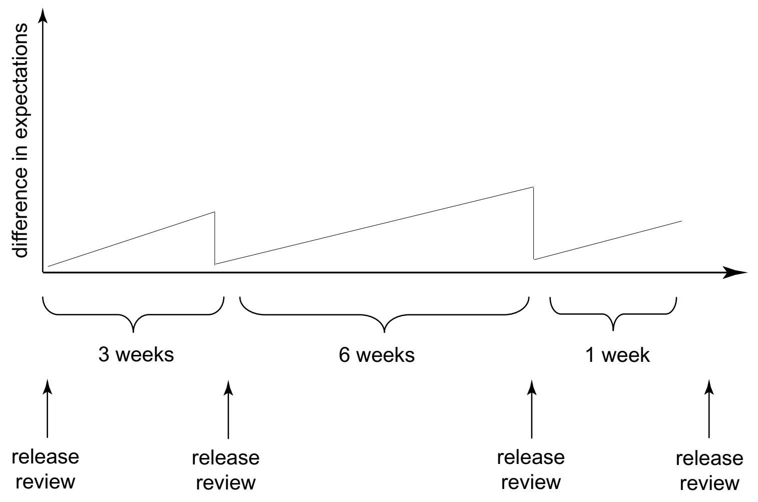


Exhibit 15 Managed Difference in Expectations

Development and the users were in a vicious cycle. The developers and the DSM managers had the most closely aligned views of a proposed software application at the beginning of a project. Of course, the application was not defined, so everyone could imagine anything they wanted. The development group would work through an initial phase of the DSM-DM. During this phase, the user's imagination and the vision in the developer's eye would evolve independently. At the phase review, it would be clear how expectations had diverged. In conversation, everyone tried to explain his or her current understanding of the product. At the end of the phase review, expectations would be closer, but never close to perfect consensus.

The process was frustrating and many coped by practicing avoidance. They knew the communication process was imperfect and wearisome, so a manager would often delegate the responsibility of participating in the phase review, often with good reason; the person chosen to participate was usually closer to the operation than the manager, perhaps with a better understanding of the technology. But always, the delegate had less authority to make decisions. Phase reviews later in a project often required several meetings as the delegate needed time to return to her or his manager to discuss options and then return some days later.

Both developers and users find it easier to maintain a high level of understanding with the advent of incremental development. Most communication occurs on a very informal level with users commenting on the latest change, knowing that their concerns will be addressed in a release in the near future. Both users and developers have reported that the issues being addressed are never big, emotional issues, but rather a series of smaller negotiations (see Exhibit 15).

During one discussion, a regional manager recalled how contentious the process was years ago and said, "Now we discuss problems and errors so quickly after they are detected, we don't even call them errors and problems."

The biggest issue now is the prioritization of requests. Users want the changes and enhancements as quickly as possible, but there is a limited number of developers. So, decisions have to be made as to which requests are addressed in what order. The CIO posts the current development plan (list of tasks, who is working on them, and the task's status) on the department's intranet site; this has been a big help in facilitating the communication process. The development plan is also connected to an online discussion conference where users and developers can exchange ideas and issues between physical meetings (which are always difficult to schedule).

Chapter 9

Funding and Economic Return…

Or Paying the Way

Your goal is to add value to the enterprise. The evidence of value is in the support you build from other divisions of the enterprise, not in "bringing projects in" on time and under budget. The funding model for software development must be tied to the value of the result. Small is good. As software development efforts get bigger, two destructive processes occur. The obvious effect of size is increased risk. The more insidious effect is that objectives are twisted from adding value to "getting done."

Organizations do not need projects, they need systems.

—Robert Block

Funding as Risk Containment

The risk depicted on the *y*-axis in Exhibits 1 through 4 corresponds to the probability of failure of a development effort as things get "big." The "big" scale on the *x*-axis can be the time horizon. Development efforts that plan long periods of time before real demonstrable results are delivered run the risk of solving the problem that existed at the start of the period and not knowing that the problem has changed until after significant resources have been wasted.

The "big" scale on the *x*-axis can also be the size of the human effort. Most of us have had experience working in large organizations. You recognize

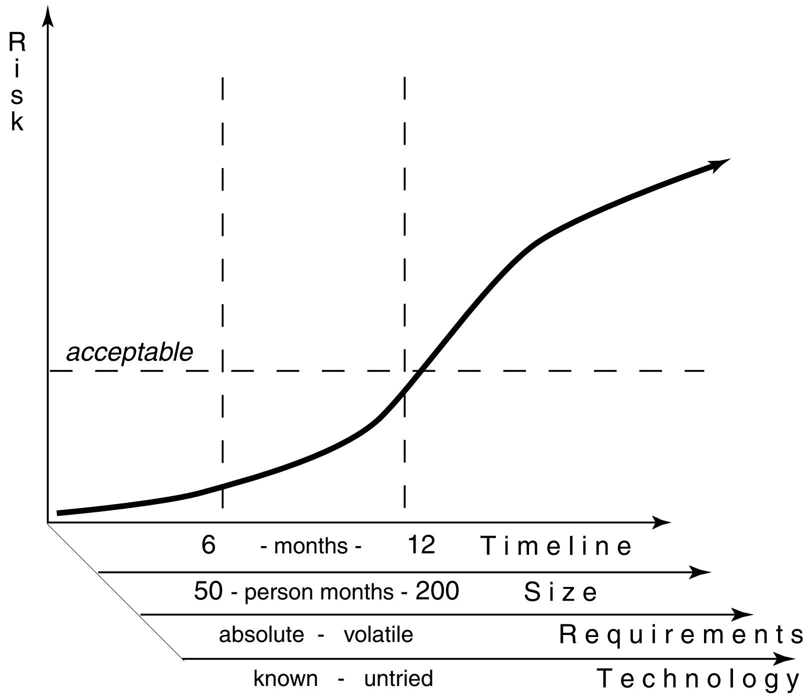

Risk
acceptable
6 - months - 12 Timeline
50 - person months - 200 Size
absolute - volatile Requirements
known - untried Technology

Exhibit 1 Risk versus "Big" Timelines

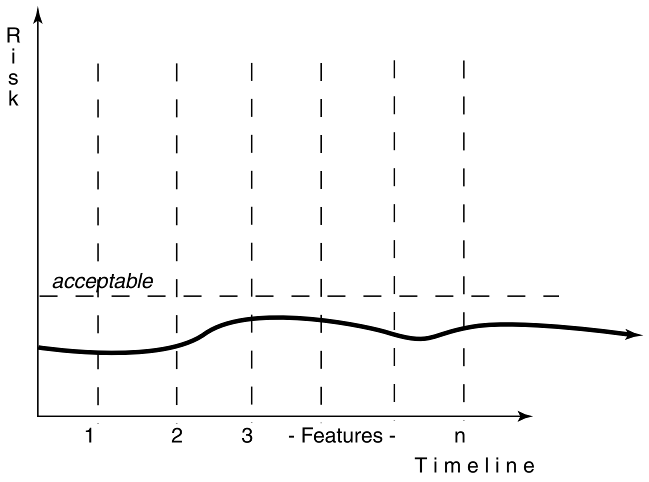


Exhibit 2 Risk versus "Big" Human Effort

and accept that as the size of the group increases, more resources must be expended just coordinating work. Colleagues working in large organizations report they spend about 30 percent of their time in planning meetings or other coordinative activities, while colleagues working in small development shops report that less than 10 percent of their time is spent in such activities.

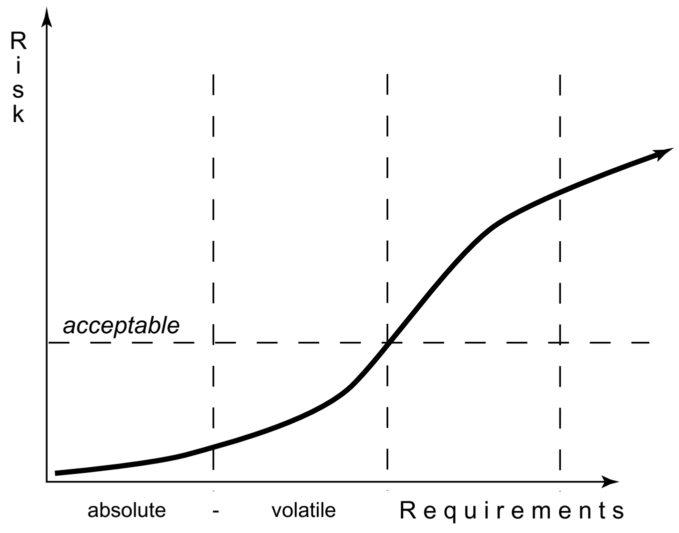


Exhibit 3 Risk versus "Big" Requirements

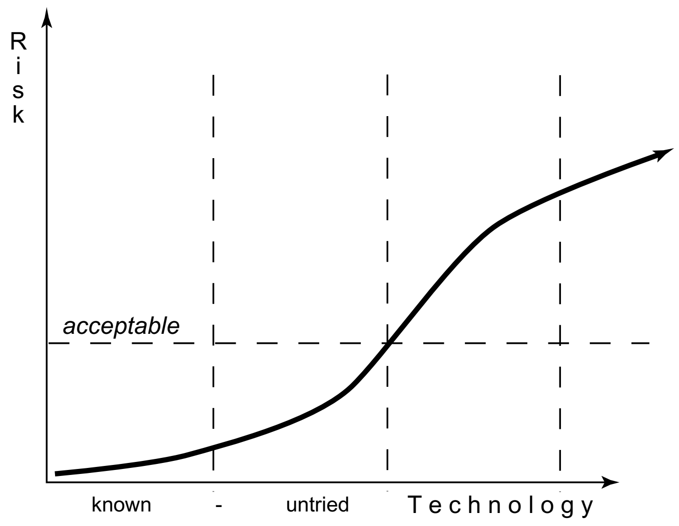


Exhibit 4 Risk versus "Big" Technology Change

Both groups speak of these meetings with approximately the same degree of disdain.

The risk curve looks the same for graphs where the *x*-axis scale represents the uncertainty of the requirements. The risk of failure is very low in domains where all users know with certainty and agree on the system functions and information definitions that will satisfy their requirements. I have never seen such a user domain, but in theory it could exist. More often, the users do not know with any degree of certainty what will best serve their needs. In this more common case, small incremental cycles provide room for review and critique before a large expenditure of resources is made on building to the wrong set of requirements.

The "big" scale to which many development shops fall prey is the allure of big technology. Most developers like to think of themselves as researchers on the forefront of advancing technology innovation. Given the choice between old and tired technology and new and promising technology, most developers will opt for the latter path. The farther out on the new technology scale we go, the higher the risk of failure because the right end of the axis increases the number of unknowns. The most successful shops are those in which developers are excited about solving problems — even if that means using two-generations-old technology. The shops with the worst reputation (and the least support for funding) are those perceived as being technology-driven hotshots.

In all of these scales, the number of unknowns grows significantly as you move out the *x*-axis. Users are unsure of their requirements and developers are unsure of what it will take to build a system that is tenuously described. But confidence is usually high, even if specifics are lacking.

Increased risk is largely due to the uncertainty inherent in larger scales. This truism is recognized by older prototyping approaches to building software and by the newer Agile and Extreme methods. These development philosophies recognize that the probably of success is improved with small cycles and many opportunities for user and other stakeholders to provide critical feedback.

The ratio of "big" to risk also has a significant effect on the funding process. "Big" kicks off a vicious cycle. If there is a belief that system enhancements occur in long cycles and with great effort, there is a tendency to expand the scope of their requests. This expands the time horizon, which increases the perceived risk. It is reasonable to be concerned about resource requirements for a development effort with projections of a long time period and uncertain future benefits. As managers, we need to be concerned about setting limits on the risk the enterprise will assume.

Most enterprises behave as if they were following a simple algorithm.

```
if (the money spent is less than $x) and (the elapsed time is less
than n months) then
   the resulting software will not be valuable
otherwise
   the resulting software will not be valuable
end if
```

With long time frames and many people negotiating for influence, agreements become heavy, formal, and less flexible. By necessity, the objectives are fuzzy. After all, we have added as many requirements as we can to the proposal and we know that many will not be implemented for a long time. But, the money and time thresholds are firm. So, it is not surprising that all managers involved begin to focus on the numbers. The goal becomes that of matching the actual and projected budget and matching the actual and projected time estimates rather than providing valuable software solutions.

The communication on the delegate channel (the interaction with your boss) fills up with questions about how we can be sure we will not go over budget. This means less space on the channel for discussion of improving the framework and prioritizing objectives.

Communication on the collaborate channel (interaction with your peers) concentrates on watching the developer's timesheets and creatively interpreting requirements to be sure that deadlines can be met. The discussion is not: "Are we getting better?" but: "We will not meet the time/money goal, therefore, what can we do to catch up?"

> "Necessity is the mother of invention" is a silly proverb. "Necessity is the mother of futile dodges" is much nearer the truth.
>
> **—Alfred North Whitehead**

Communication on the service channel (the interaction with users) often deteriorates to conflicting objectives. The development manager is asking questions about stabilizing requirements so the money and time thresholds will not be exceeded. User management is asking questions about how functions and definitions can be modified and made more relevant to current needs (as opposed to the needs imagined when the effort began).

Dynamic and iterative development management can help replace the previous algorithm with a more constructive one:

```
if (the last delivered feature was valuable to the user) and (the user
turns to me for greater support) then
   my development effort will get increased funding
otherwise
   the organization will begin to question my value proposition
end if
```

The variables in this equation are not quantitative. You will need to promote your contribution. If users feel you are adding value, promoting your success will not be difficult.

Funding the Perpetual Effort

Many managers with whom I have worked have fallen into the cycle of "big." One manager described it as a game show where the development group is asked to bid on cost and time projections — "I can name that tune in three notes." Development managers are bidding for approval by claiming, "I can deliver that service for $1.2 million in 11.5 months."

> To measure up to all that is demanded of him, a man must overestimate his capacities.
>
> **—Goethe**

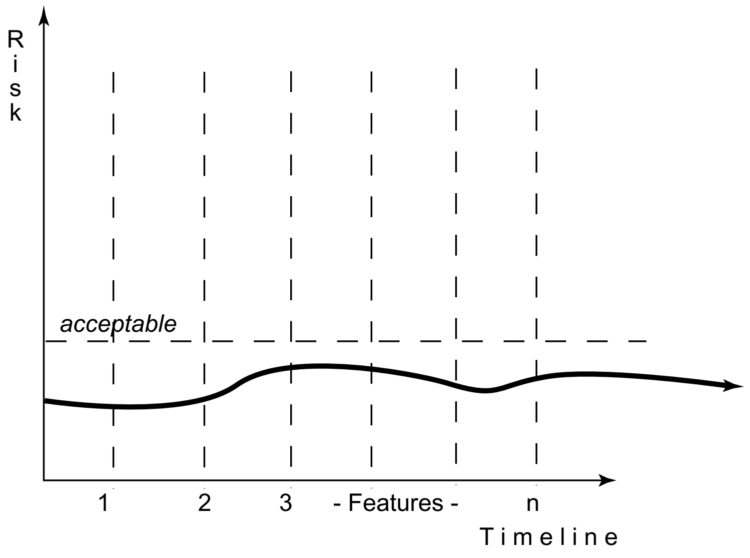


Exhibit 5 Small Steps, Small Risk

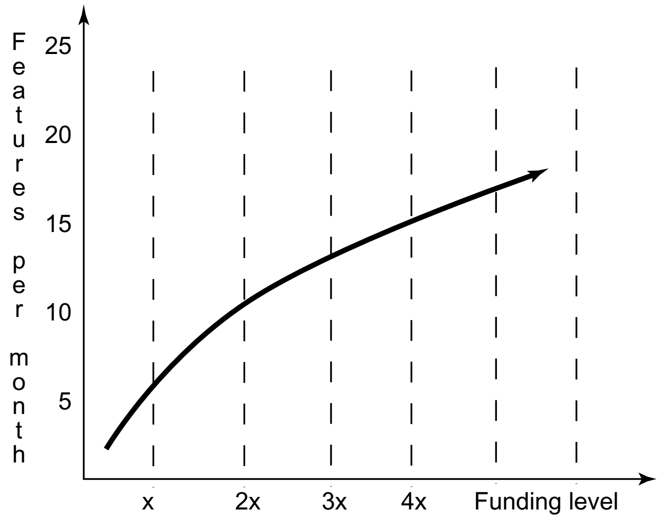


Exhibit 6 Funding versus Service

Funding for the perpetual effort is based on continuous development, keeping a short time horizon and maintaining risk well below acceptable levels (see Exhibit 5).

If there is a belief that new features and enhancements have a short cycle requiring smaller effort, the funding process becomes an operations budget directly associated with benefit. If the development organization is delivering five features per month at the current funding level, doubling the funding level should yield eight or nine features per month. The ratio is not linear (see Exhibit 6). As development organizations increase in size and applications increase in features, productivity drops some due to the increased requirement for inter-team communication and coordination.

The justification for increased funding should be the backing and testimonials of the users. If the service provided by the development organization is visible and positive, support for a steady funding stream is enhanced by highlighting the work completed in the past and having user support for future plans.

> Change is one thing, progress is another. "Change" is scientific, "progress" is ethical; change is indubitable, whereas progress is a matter of controversy.
>
> **—Bertrand Russell**

The curve shown in Exhibit 6 implies a nonlinear relationship between funding level and delivered features. It would be nice to think that if I double the funding to the development effort, I double the productive output. However, the property of "big" comes into play again. As the development effort grows and more resources (both human and hardware) are applied to the creation of productive applications, a larger portion of the resources has to be applied toward the coordination of the development effort itself. As managers, we need to be vigilant and watch for the point where the slope of the curve is so shallow that increased funding does not yield sufficient benefit.

> I have never seen a development organization suffering from this problem. This is a problem most managers would love to have.

Paying for Asset Development

The features and services built by the development organization are capital assets. But unlike software's physical counterparts (e.g., plant and equipment), software development costs are difficult to estimate and benefits are difficult to quantify.

> I find it interesting that all the cost and size models show that software costs and schedules are very consistent across projects. But the analysis is always in hindsight. After the project is completed, the numbers line up. Before the project, it is anybody's guess.

It is far more comfortable to base funding and budgeting decisions on the body of work. The development plan (see Exhibit 7) is a valuable record of past and pending work that users are demanding and the development organization is providing. You should highlight the direct connection between the resources going into system development and the benefits received by the users. You should keep the discussion on all your communication channels

Exhibit 7 Development Plan

Task ID	*Focus Element*	*Element Count*	*Description*	*Priority/ Sequence*	*Assigned*
A	—	—	Add customer order status	Done	Blue Team
B	—	—	Add customer credit query	Done	Green Team
C	—	—	Enhance product pricing formula	1	Blue Team
D	—	—	Replace inventory valuation method	2	Green Team
E	—	—	Add product inquiry	3	Unassigned
F	—	—	Convert customer database	4	Unassigned
G	—	—	Replace order entry sequence	5	Unassigned
H	—	—	Link credit history to order qualification	6	Unassigned
I	—	—	Pull directly from supplier catalog	7	Unassigned

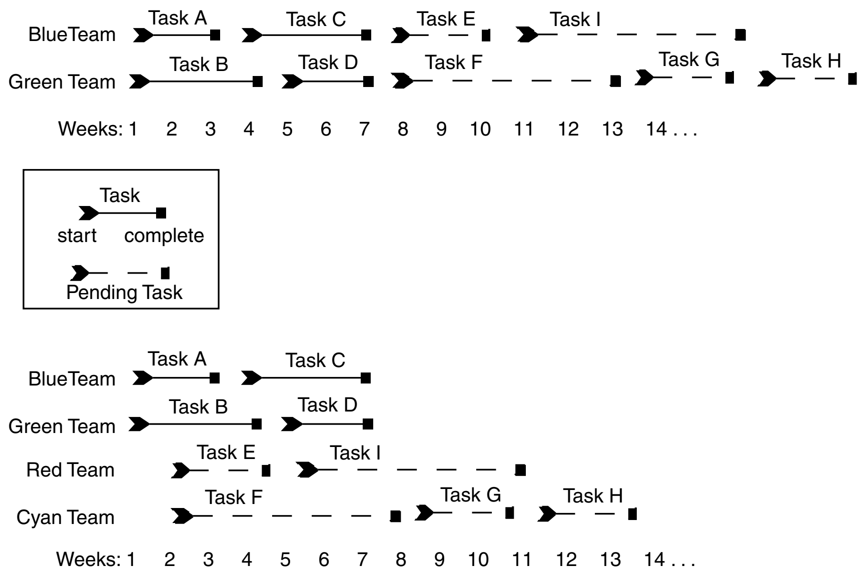


Exhibit 8 More Resources, More Service

focused on the proposed services and the resources applied to the tasks that yield direct, demonstrable benefit.

The development plan lists the pending services and who is assigned to perform them. There are always more "pending" tasks than "assigned" tasks. Tasks completed last year will be enhanced this year. It is the nature of the beast. The funding model should be as natural as the work. We have a given amount of resources, and we (all affected managers) sequence and prioritize the work as best we can. Developers work in an environment in which their goal is to produce sustainable solutions with the most effective technology available. Developers work in direct response to needs in the enterprise; if we want more service, we buy more resource (see Exhibit 8).

The upper and lower portions of Exhibit 8 depict two options for scheduling Tasks A through I. The discussion on the service and delegate channels should be prioritizing the pending tasks and weighing the possibility and benefits of forming additional teams.

Be prepared for the issue of "change." Every project in which I have been involved has faced the argument that software built now will have to be changed at some point in the future when related elements of the repository are addressed. I remember comments like: "The product table will have to be redesigned when we change DBMS vendors next year — so why take time to address Task C now?" Or, "They're hiring a new Director of Marketing sometime in the near future — so let's bundle that into Task J."

> Perfect accuracy of thought is unattainable, theoretically unattainable, and undue striving for it is worse than time wasted: it positively renders thought unclear.
>
> **—Charles S. Peirce**

A constructive response is to describe the following scenario:

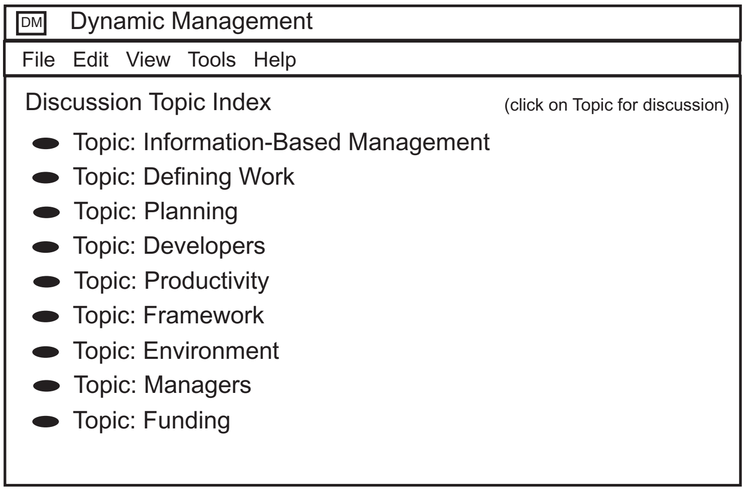
- Today, elements of the system are changed.
- Tomorrow, the user identifies a valid need that involves the recently changed elements.
- It is ridiculous to say, "We made a change to that element yesterday so we can't address your need today." The decision to dedicate resources to fulfill that need is not affected by the fact that related elements have recently been changed.
- It is equally ridiculous to say, "We may have to change that element tomorrow; therefore, we can't address your need today." The decision to dedicate resource is not affected by the possibility that future needs might require change to those same elements.

If the need exists and the enterprise decides it is important enough to take priority over other pending tasks, it is the right decision to start work on the newly identified need.

> Always have a bias toward action.

Applying Dynamic Management

Review the last project budget you approved for your department (see Exhibit 9). Consider the role of the budget in your development effort. Try to remember the criteria you used for day-to-day decisions. How prominent was the desire to work to the budget numbers? Try to assess how the decisions might have been different if the criterion had been to maximize service to the user.



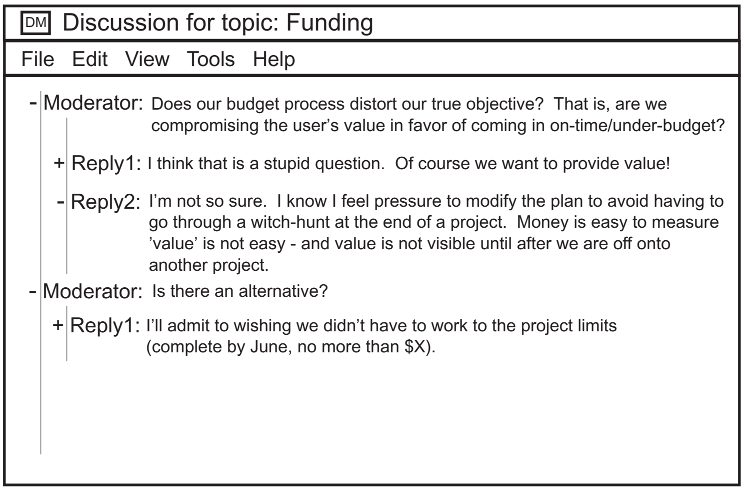
DM Dynamic Management

File Edit View Tools Help

Discussion Topic Index (click on Topic for discussion)

- Topic: Information-Based Management
- Topic: Defining Work
- Topic: Planning
- Topic: Developers
- Topic: Productivity
- Topic: Framework
- Topic: Environment
- Topic: Managers
- Topic: Funding

Exhibit 9 Online Discussion Index



DM Discussion for topic: Funding

File Edit View Tools Help

- Moderator: Does our budget process distort our true objective? That is, are we compromising the user's value in favor of coming in on-time/under-budget?
  + Reply1: I think that is a stupid question. Of course we want to provide value!
  - Reply2: I'm not so sure. I know I feel pressure to modify the plan to avoid having to go through a witch-hunt at the end of a project. Money is easy to measure 'value' is not easy - and value is not visible until after we are off onto another project.
- Moderator: Is there an alternative?
  + Reply1: I'll admit to wishing we didn't have to work to the project limits (complete by June, no more than $X).

Exhibit 10 Beginning Dialogue

Most of the time, the differences are minimal and no real dissonance exists. But if a significant number of decisions were driven by a need to meet budget and schedule constraints, there is room for improvement.

Discuss the budget you have just analyzed and your memory of events with your lunchtime management group. See if others have had similar experiences. Then discuss the possibility of shifting to a more continuous and operational funding model (see Exhibit 10).

Chapter 9 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

Exhibit 11 Initial Memo to the CEO

To: CEO

From: VP of Marketing

Subject: Support from Applications Development

We need more support from applications development. We have submitted many requests for new and updated systems. While it appears the people within applications development have the interest and the expertise, they clearly do not have the time. We are told that they are busy with updates to the financial applications and that our requests will be addressed when resources are available. In my conversations with the CFO, she claims she has no more money to hire more developers and that her applications will continue to take precedence.

This is untenable. We would prefer working with the internal development group but there are third-party vendors with software systems that will fill most of our needs. I need to know if I should continue to investigate outside sources for system support or if there will be development resources available that I can count on.

cc:

CFO

COO

Director of Applications Development

International Directors

DSM had more requests for system support than they could handle. Marketing wanted customer support for the regional managers, operations wanted systems to aid the consultants, and the financial people had a long list of enhancements to internal accounting systems. With the Director of Applications Development reporting to the CFO, the financial applications were getting preferred attention.

The VP of Marketing sent a memo to the CEO expressing his frustration, as shown in Exhibit 11. He also copied the memo to the other executives in DMS because he knew they were feeling the same frustration and wanted to ensure the debate was enlarged to include all parts of the DMS enterprise.

The COO and directors of the international divisions replied with similar requests. They all said they would prefer working in-house, but that the level of service was not adequate to support the company's growing needs. The CFO requested more money to hire more developers and upgrade the development tools. The CEO was a little confused by all the discussion but relied upon his officers, agreeing to amend the budget and increase the investment in the systems development area.

The CEO's confusion came from the fact that everyone was talking about applications development as if the systems were capital assets — like they were requesting more plant and equipment, but the company was accounting for the development area like a clerical function. It was a cost center. Usually, everyone tried to minimize the money consumed by cost centers, but here, there was almost unanimous agreement that the company should “invest” in the division. The CEO wondered how he would know if the “investment” yielded a positive return.

The CEO had several conversations with the Director of Applications Development. He ask the Director how he felt about the current state of affairs. The Director reported that it was awkward. He wanted to be able to work with the other divisions on marketing and customer support but, until recently, was only able to focus on the financial systems. They talked about the possibility of moving the development function into the marketing area or have several development groups — one for marketing, one for finance, one for operations. They agreed that it would be difficult to coordinate and that it seemed like a duplication of effort. Both men thought about the idea of applications development reporting directly to the CEO, but neither actually verbalized it. The CEO was uncomfortable with the technology and, in general, it seemed like an odd notion.

The CEO explained to the Director that he was increasing the funding for applications development and that he was sure the CFO would support the development of nonfinancial systems requested by other divisions in the company.

Over the next year or so, the CEO saw that various projects were initiated. He reviewed the accounting of the money being spent on these projects and convinced himself that things were moving in the right direction. But direction was not always without its problems.

The CFO was very uncomfortable managing a division that was increasingly answering to other managers. She recognized that the Director of Applications Development was in a difficult position as he tried to coordinate development in support of many parts of DSM. Increasingly, conflicts developed, such as the one between finance and marketing. A project to develop customer relations information for the regional managers had been approved. It was expected to take six months to complete. The CFO accepted a delay in a proposed upgrade to their customer credit system, expecting the work to begin after the marketing system was completed. After about five months, it was clear that the marketing system was not going to be completed on time and the capital management system would be delayed even further. The Director of Applications Development suggested that some of the developers working on the marketing system be reassigned to start the customer credit project. This meant that the marketing system, already behind schedule, would be even later.

The CFO resented having to spend time on this conflict. From her perspective, she was being generous to support the marketing systems at all, and now to have her generosity repaid with delays in critical systems was difficult to take.

The Director went to his developers and asked for extra effort to get the marketing system “out the door” as soon as possible. The developers responded by putting in a significant number of overtime hours. They also were liberal with the interpretation of requirements; after all, it was clear that the goal was on get “the system” out the door on time. The exact definition

Exhibit 12 Memo from the Director of Applications Development

To: CEO

From: Director of Applications Development

Subject: Project prioritization

The Applications Development department needs more information about the strategic contribution computer applications are making to DSM. We have a long backlog of system development requests. Our effort to prioritize the requests requires more information than we have. The decisions we make about which projects take precedence will never be optimal without input from all affected DSM divisions.

I request that a steering committee be created with representatives from Finance, Marketing, Operations, and International Divisions. Their charge would be to approve and prioritize development requests and act as overseer and facilitator of relationships between the development group and its users.

cc:

CFO

of “the system” had always been nebulous. So the fact that extensive error handling was not implemented was technically not a violation of requirements because it was never explicitly stated. Cutting corners on the rigor of data interfaces seemed justified because it allowed them to deliver “the system” on time.

They succeeded by the measurements that were most prominent. The system was put into production only two weeks late. The marketing people were a little disappointed that some of the features they thought would be there were not and that the training and conversion support they thought they would have was rather sketchy. The developers were happy to have their weekends back and they began working on the customer credit system.

As time went on, marketing began sending more and more maintenance requests. They had discovered that it was easier to get maintenance requests in the queue than “new development” requests. But the Director noted that some of the maintenance requests were larger than the original new development proposals. As he assigned people to deal with the maintenance jobs, there were fewer developers working on the new systems.

The Director of Applications Development knew that the situation was not going to get better. He reasoned that as long as the managers requesting services were outside the decision-making process, there was going to be dissatisfaction with any decisions made. He contacted his boss (the CFO) and the CEO, suggesting the creation of a steering committee (see Exhibit 12).

The Director had a conversation with the CEO in which the director expressed his hope that the steering committee could exercise a degree of authority in mediating the decision-making process. The CEO agreed and called the first meeting of the committee where he established their mission. The first few meetings of the steering committee were attended by the executives themselves. Their initial work seemed promising; they made decisions about which of the

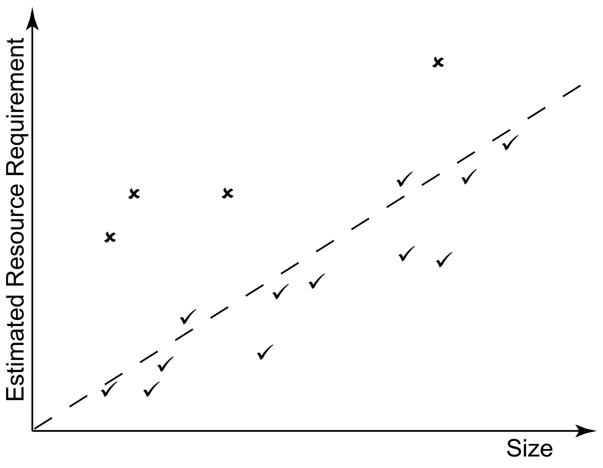


Exhibit 13 Expected Project Funding

development requests should have priority. They were even successful at merging several related requests by identifying overlaps in the subject matter and assigning subject matter experts from different divisions to work jointly on an integrated system.

While the steering committee did help coordinate the expectations of the users, there were some unanticipated consequences; most significantly, the size of the requests became larger. The steering committee added a layer of formality to the process of selecting projects. For all members of the committee to vote in favor of a proposal, the proposal had to be significant. Smaller projects were viewed as taking valuable resources that should be directed to more "important" work.

The Director and one of his senior analysts were talking one day after a steering committee meeting. The analyst wondered aloud about the trends they had seen in the project approvals. He drew a chart representing the size of a proposed project against the estimated resource requirements. "You would expect a linear relationship, wouldn't you?" asked the analyst (see Exhibit 13). "Bigger projects require more resources. But you would expect the approved and funded projects to cluster along a threshold, with proposals perceived by the steering committee as too costly being rejected."

"This assumes that perceived benefits are proportional to size," the CIO added. "Right," continued the analyst. "Small project — small benefit. Big projects — big benefits. But that is not what is happening." The analyst started another chart with axes labeled "resources" and "size," but this time he began marking recent project approvals.

The CIO and the analyst began listing the proposals recently considered by the steering committee; marking an "x" for rejected proposals and a checkmark for approved proposals, positioning the mark on the graph (see Exhibit 14) where it represented their perception of the project's relative size and estimated resource requirement.

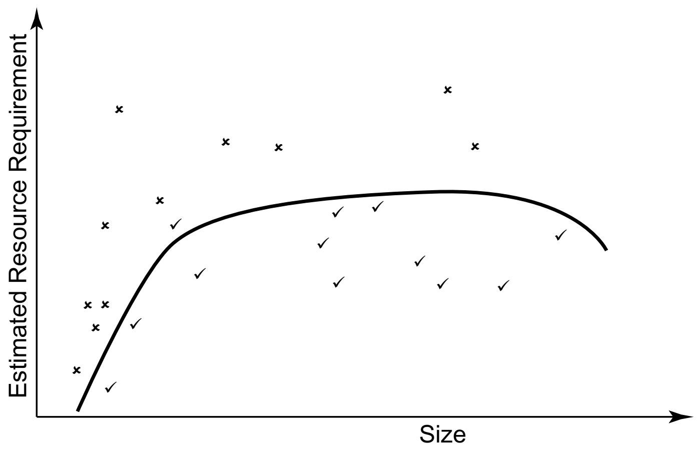


Exhibit 14 Actual Project Funding

- More smaller projects were rejected than larger ones.
- Rejected proposals were the ones with the higher resource estimates.
- As projects got larger, the resource estimates seemed to be proportionally smaller.

"This last point is insidious," commented the Director. "We are under pressure to define proposals that will satisfy the largest number of demands. If we are honest and accurate with estimates, they appear too large. Large projects are nearly impossible to estimate accurately, so there is always pressure to agree to lower resource estimates. And the big projects with lower estimates are the only ones that get approved. Such a deal!"

Chapter 10

Leadership by Consensus…

Or if You're Going My Way

Dynamic system development requires flexible and adroit decision making. Your network of colleagues forms the mechanism for strategic planning. Leadership is required. But most of us in development management are not in a position to lead from a position of organizational power, so we lead through influence, support, and consensus.

> A reasonable change of the world cannot be instrumented by pure reason.
>
> **—Friedrich Dürrenmatt**

Decision Councils

People in most enterprises recognize the need to bring different expert perspectives to bear on the problems of determining the direction of software development and prioritizing development proposals. Because developers move through the development tasks quickly, you need input from the rest of the enterprise that is timely and responsive.

Unfortunately, the most convenient means of instituting this decision-making process is to appoint a steering committee. These committees are typically populated with representatives of different levels of management among different organizations of the enterprise. One example looks like this:

- CIO (or delegate)
- Director of Systems Development
- Senior systems analyst
- VP of Marketing (or delegate)

- VP of Production (or delegate)
- VP of Finance (or delegate)

The steering committee meets regularly to review software development proposals, discuss problems with current efforts, and plan future direction.

While this form of decision council is common, it is not without its problems. Often, the senior representatives have other responsibilities and send delegates on their behalf. If the delegate does not have the authority to make decisions, the meeting becomes a discussion session with decisions being postponed until the senior members can attend. Also, seldom do issues being discussed affect all members present. A proposal to enhance Marketing may not require the expertise of the delegates from other operations. The unaffected parties are wasting their time while the interested members of the steering committee are dealing with the item.

As a result, interest and commitment to the process becomes weak and the lead-time for decisions becomes long. You, as a development manager, would be better served if the function of the decision council were distributed and more dynamic.

A combination of electronic conferencing and one-on-one or small group meetings can serve the purpose of the steering committee without the long delays. Membership in the conference is still important, but the number of participants can be larger because there are no scheduled meetings that require all individuals to coordinate their calendars. The wider membership is good because you need the expertise of many individuals — but not all of them all the time.

The basic structure of the electronic conference is depicted in Exhibit 1. The conference requires some facility on your intranet or file servers, allowing you to post an item to be considered. The conference must allow for a threaded discussion so participants can post comments related to the question at hand and read the opinions of other interested participants. This repository allows any interested manager in the enterprise to view and monitor items being considered without requiring participation in physical meetings. The dotted vectors in the figure represent person meetings (e.g., phone call, lunch meeting, coffee break, conversations in their offices).

The items in the conference are proposals for development work that must be prioritized, suggestions for changes to the environment that must be approved, or strategic considerations that need input and discussion. When you receive an item, you post its description on the conference and announce (usually via e-mail) that an item is open for debate. The posting must include your initial understanding of the significance of the item (i.e., why it is important to the group).

You then decide who the item directly affects and whose input will be critical for thorough consideration of the item. You must discuss the item with these individuals directly to gain an understanding of their perspective. Ask them to post their views on the conference to make sure their views are voiced and so others can benefit from their ideas.

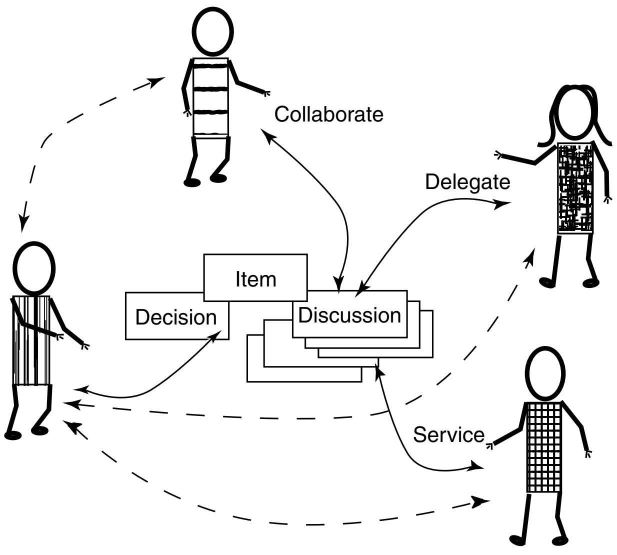


Exhibit 1 Structure of Electronic Conference

You need to synthesize the information from the conference and from one-on-one conversations as quickly as possible and arrive at a consensus — or what you believe to be a consensus. Post this conclusion on the conference, asserting that it appears the group is in agreement and asking that participants reply if this is in error.

The process continues. More items are received and posted; you lobby to ensure that people critical to a given decision are notified and their opinions are shared. You advance the decision that emerges from the discourse.

Thought allied fearlessly to purpose becomes creative force.

—James Allen

Leadership through Criticism

The dynamic and largely asynchronous process outlined above requires perpetual and ongoing effort on your part. Most of the development managers with whom I have worked have had very little power, yet their positions require them to orchestrate significant, far-reaching decisions. The participants in decision making had far more managerial power. The successful managers wheedled influence by keeping issues in the open, helping the players succeed, and always acting in the best interest of the enterprise.

An effective technique for managing the communication channels and keeping valuable people involved in the strategic support for software development is to offer each person many chances to criticize the development plan and proposed decisions. If you post a task definition and invite participants to comment, you typically get a couple of quick, well-meaning but poorly considered replies. If you post a message asserting that it is your understanding that the group is in agreement that a task should be approved and that its priority will cause other tasks to be delayed, you will get many comments with real and valid concerns. Of course, the issues raised by the replies are often contradictory and mutually exclusive, but you get valuable information that you can use to forge the modifications necessary to keep the development effort moving. The issues raised must be addressed, but having them out in the open gives you the possibility of completing constructive work.

The conference evolves into a shared memory where the current status of the development plan is visible and proposed changes can be discussed in the context of current service and other pending requests. The conference is often a lightning rod for reaction (positive and negative) to changes in the status of the development effort. A manager is likely to speak up when she discovers the product pricing function she requested is going to be delayed because another task is going to be assigned a higher priority. A manager is not likely to pay much attention if the proposed higher priority task is presented without stating the consequences of approving it.

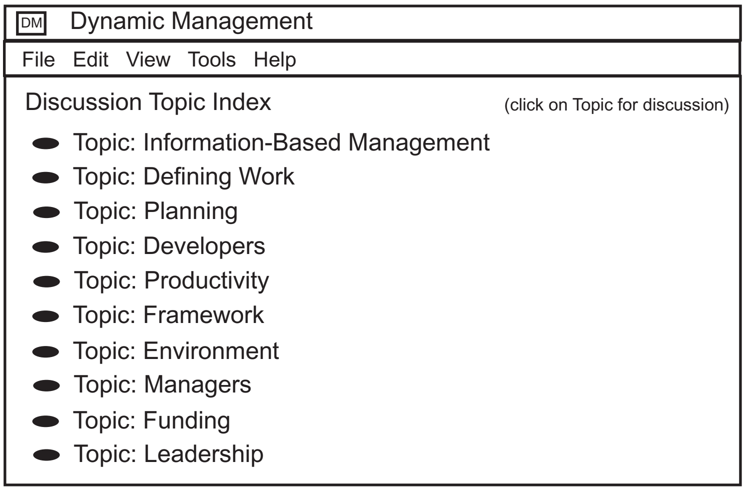
The development manager must then invite criticism and subsequently use this information to present alternatives. Tasks can be switched in sequence, task definitions can be redefined so that different combinations of elements are created to satisfy the major player's concerns, more resources can be added, or some users can be convinced that the new priority makes sense. The development manager must keep all the goals visible and resources fairly applied for the benefit of the enterprise.

> The strength of a man's virtue should not be measured by his special exertions, but by his habitual acts.
>
> **—Blaise Pascal**

Need for Responsibility

You as the development manager must take on the responsibility of soliciting input and facilitating the discussions. There is typically no power or reward in this position, but the success of the development effort depends on keeping the people affected by the development effort involved in decision making. If you see one participant dominating the conversation, you have to intercede to be sure the discussion is always fair and just.

All participants must sense that the discussion is open and that you do not have hidden agendas. This is critical to maintaining the trust of both users and your superiors. The fact that enhancements and new features are added quickly and regularly is also critical to maintaining the communication channels. We are more likely to pay attention and be focused on tasks that deliver service weekly or monthly than we are to pay attention and be focused on a project that will probably do something for us in the next year or so.


Dynamic Management

File Edit View Tools Help

Discussion Topic Index (click on Topic for discussion)

- Topic: Information-Based Management
- Topic: Defining Work
- Topic: Planning
- Topic: Developers
- Topic: Productivity
- Topic: Framework
- Topic: Environment
- Topic: Managers
- Topic: Funding
- Topic: Leadership

Exhibit 2 Online Discussion Index

Applying Dynamic Management

Investigate the possibility of setting up an electronic conference on your organization's intranet. With the experience you have gained from discussing topics in this book, you may be in a position to use the same communication method in defining and prioritizing development plans (see Exhibit 2).

Talk to the managers in your lunch group about the idea of posting development plans in this public area. Start by using e-mail to send out calls for participation in strategic decisions. Identify the set of people who must be involved and meet with them personally. Suggest the idea of using a Web conference for sharing information and gathering opinions.

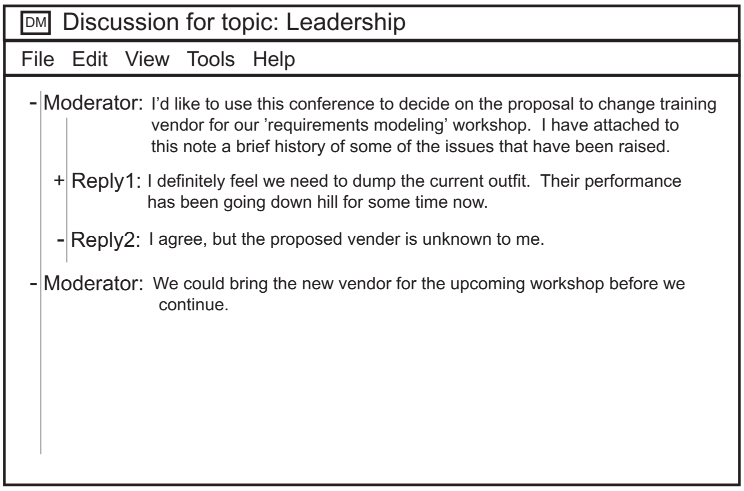
The first issue should be small enough to address in one week. Post a summary of an issue and invite participation. As soon as you think a consensus has formed, post a message suggesting what the group has decided and that you will accept that result unless someone raises new issues. Take note of the responses and talk with the participants to learn what they liked about working through the conference (see Exhibit 3).

Chapter 10 Case Study Excerpt

The following narrative is a set of excerpts from the full case study in Appendix G relating to the topics in this chapter.

As the use of technology within DSM became more integrated, the number of requests for additional computer support grew (see Exhibit 4).

The communication process seemed to improve over the first years of the committee's existence. With representatives of all major domains of the company, decisions were more open and people were aware of any conflicting



- Moderator: I'd like to use this conference to decide on the proposal to change training vendor for our 'requirements modeling' workshop. I have attached to this note a brief history of some of the issues that have been raised.
  + Reply1: I definitely feel we need to dump the current outfit. Their performance has been going down hill for some time now.
  - Reply2: I agree, but the proposed vender is unknown to me.
- Moderator: We could bring the new vendor for the upcoming workshop before we continue.

Exhibit 3 Beginning Dialogue

Exhibit 4 Increased Requests for Computer Support

To:	DSM Officers
From:	CEO
Subject:	Applications Development Steering Committee

I am forming a new steering committee to address the growing demand for a computer applications system. All divisions of the company have identified the need for new applications supporting both internal functions and our clients. It is clear we cannot accommodate all the requests at once, so a better method of prioritizing application requests is in order.

The committee will be made up of:

- CFO — (who will chair the meetings)
- VP Marketing
- COO
- Director of Applications Development
- A representative of the international divisions

I will be a non-voting member of the steering committee.

The charge to the committee is to review project proposals, review information from the requesting organization to assess potential benefits, review information from the applications development department to assess resource requirements, approve and prioritize the project requests and monitor the project's status.

The first meeting of the steering committee will be next Thursday, 10 a.m.

issues. The steering committee developed a good sense of the projects being proposed and on many occasions found ways of merging proposals from different user areas into a single project serving shared interests.

A major problem developed, almost unnoticed. The size of the approved projects became larger and larger, and the resource estimates became increasingly unrealistic. The steering committee began to view small proposals as less important than "significant requests." The small jobs only delayed and complicated the "real" work of the development group. This trend was exacerbated by the long lead-times between a project's request, a project's approval, and the completion of the work. The steering committee met only once each month. There was always a backlog of work and decisions were often delayed over several meetings. Most of the committee's time was spent on the larger proposals. People submitting project requests quickly learned that bigger proposals got preferential treatment. Smaller proposals took too long to be considered worthwhile.

After the first year, the executives began sending their subordinates in their place. This added to the lead-times on proposal considerations. Inevitably, their delegates would be faced with issues they felt they could not answer. They would return to their divisions to discuss the matter with their superiors and report back at the next meeting. This doubled the time it took to consider and act on a development request.

By the time DSM was considering the DSM-SISS system, the steering committee had become slow and unresponsive, leaving the Director of Applications Development alone to coordinate the many considerations and demands of the proposed system. The CEO stepped in to support the director. The CEO even began making unilateral decisions when the decision was on the critical path.

The close working relationship between the director and the CEO and the company's experiences during the DSM-SISS project convinced the CEO to name the director to the position of Chief Information Officer and to change the reporting relationship (see Exhibit 5). The director achieved the formal status he had earned over the years. The development organization now reported directly to an executive-level officer.

The new CIO began laying the groundwork for ending the steering committee. He worked with his developers to define a browser-based conference to access the development plan databases (see Exhibit 6). The CIO began by announcing that the current project status reports were available on the intranet and that he would not be reviewing them separately during the steering committee meetings.

Once members became accustomed to the use of the conference to reference the project status, the CIO introduced a discussion forum where project proposals were posted (see Exhibit 7). Members were encouraged to review the proposals online and post questions and comments. When the CIO determined that a consensus had been reached, he posted the summary in the discussion and asked if everyone concurred. He would then review the discussion and the result at the next steering committee meeting. Quickly,

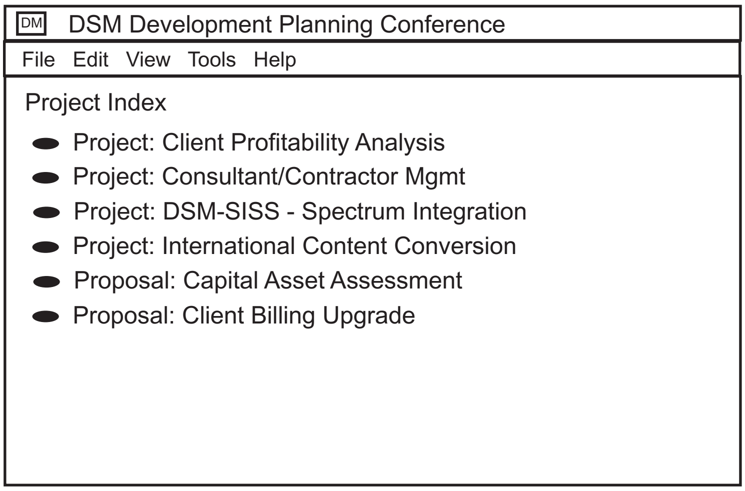
Exhibit 5 Change in the Reporting Relationship

To:	All DSM Employees
From:	CEO
Subject:	Chief Executive Officer

It is my pleasure to announce the creation of a new executive position of Chief Information Officer. I am naming the current Director of Applications Development to fill this position. All functions under the current Director will move out of the Finance department into the newly created division. The office of the CIO will report directly to me.

I, and the other officers, feel that the new position will strengthen the role Applications Development has played in the growth and success of this company. The Director has proven to be a valuable asset to the company's strategic planning and we are confident that he will continue to serve us well in his new position.

Please take the time to congratulate the new CIO on his promotion.


DM
DSM Development Planning Conference
File Edit View Tools Help

Project Index

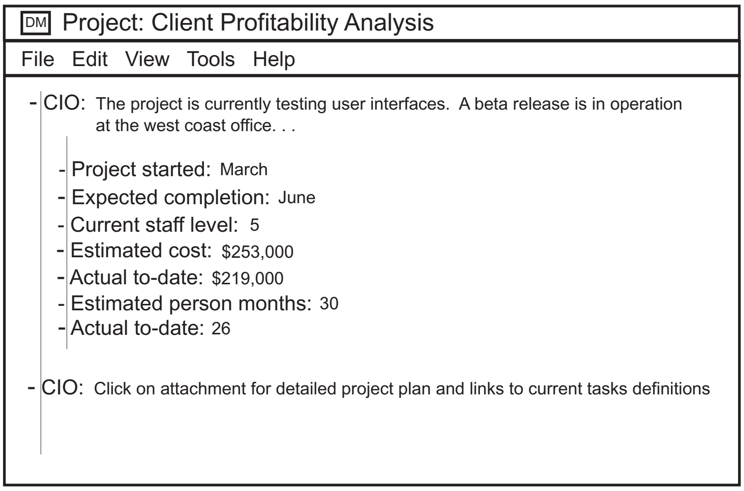
- Project: Client Profitability Analysis
- Project: Consultant/Contractor Mgmt
- Project: DSM-SISS - Spectrum Integration
- Project: International Content Conversion
- Proposal: Capital Asset Assessment
- Proposal: Client Billing Upgrade

Exhibit 6 Shared Project Repository

the members began to rely on this discussion and decision-making forum (see Exhibit 8).

When the CIO was convinced the group was functioning well using the conference, he suggested to the CEO that the steering committee meetings be canceled and that the group do all future business through the online conference (see Exhibit 9).

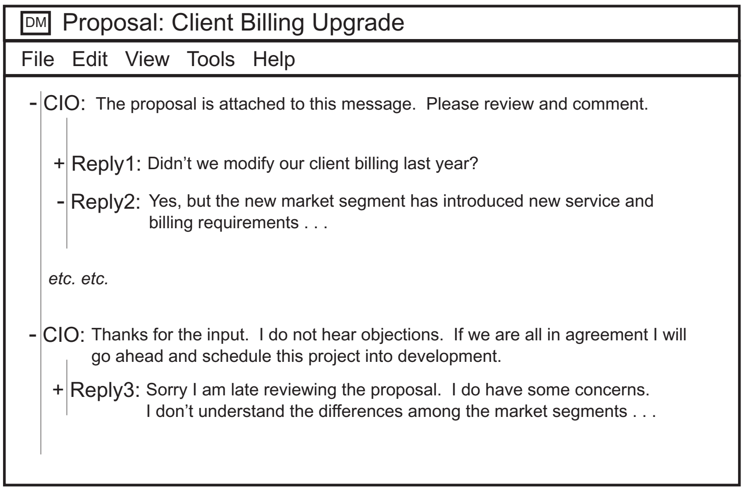
The CIO continued to meet regularly and informally with all the decision makers. But the flexibility of the online conference allowed him to conveniently invite other experts into the discussion. One proposal involved a complex interface. The CIO invited an outside expert to post information to help the steering committee assess different options.



Project: Client Profitability Analysis

- CIO: The project is currently testing user interfaces. A beta release is in operation at the west coast office. . .
  - Project started: March
  - Expected completion: June
  - Current staff level: 5
  - Estimated cost: $253,000
  - Actual to-date: $219,000
  - Estimated person months: 30
  - Actual to-date: 26
- CIO: Click on attachment for detailed project plan and links to current tasks definitions

Exhibit 7 Project Status Page



Proposal: Client Billing Upgrade

- CIO: The proposal is attached to this message. Please review and comment.
  + Reply1: Didn't we modify our client billing last year?
  - Reply2: Yes, but the new market segment has introduced new service and billing requirements . . .

etc. etc.

- CIO: Thanks for the input. I do not hear objections. If we are all in agreement I will go ahead and schedule this project into development.
  + Reply3: Sorry I am late reviewing the proposal. I do have some concerns. I don't understand the differences among the market segments . . .

Exhibit 8 Proposal Discussion Area

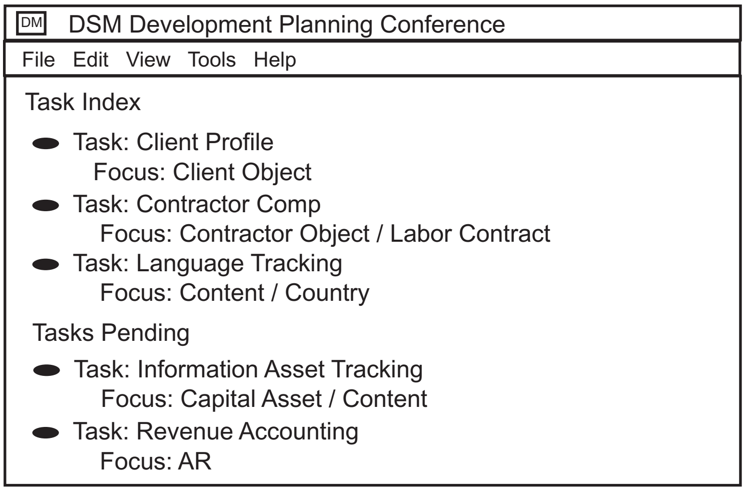
Exhibit 9 Cancellation of Steering Committee Meetings

To: DSM Officers

From: CEO

Subject: Applications Development Steering Committee

I am canceling all future software application steering committee meetings. It seems the new online conference system is working and there is no longer a need to coordinate our schedules and meet regularly for the purpose of reviewing project requests.


DM DSM Development Planning Conference

File Edit View Tools Help

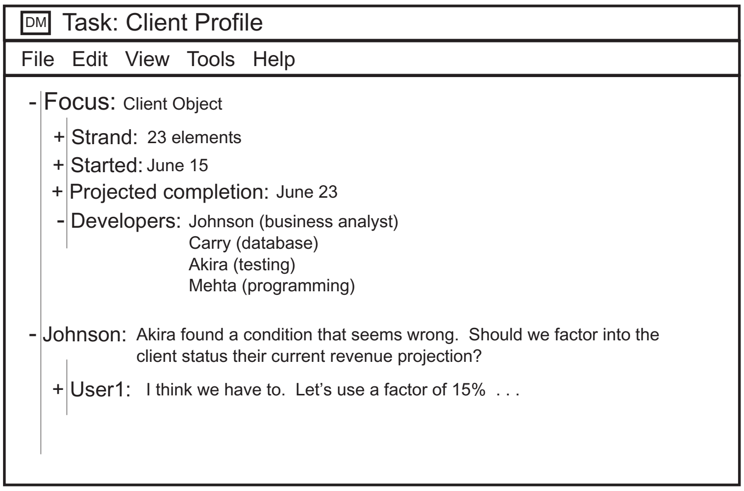
Task Index

- Task: Client Profile
  Focus: Client Object
- Task: Contractor Comp
  Focus: Contractor Object / Labor Contract
- Task: Language Tracking
  Focus: Content / Country

Tasks Pending

- Task: Information Asset Tracking
  Focus: Capital Asset / Content
- Task: Revenue Accounting
  Focus: AR

Exhibit 10 Current Task List


DM Task: Client Profile

File Edit View Tools Help

- Focus: Client Object
  + Strand: 23 elements
  + Started: June 15
  + Projected completion: June 23
  - Developers: Johnson (business analyst)
    Carry (database)
    Akira (testing)
    Mehta (programming)

- Johnson: Akira found a condition that seems wrong. Should we factor into the client status their current revenue projection?
  + User1: I think we have to. Let's use a factor of 15% . . .

Exhibit 11 Task Status

The trend toward smaller tasks and funding for continuous development continued at DMS. The use of the online development plan continued to evolve. All parties became comfortable with communicating and refining the direction of the development effort on a continuous basis.

The development plan intranet site changed from listing projects to listing tasks. The project status pages were replaced with task status pages with integrated user-to-developer dialogue (see Exhibits 10 and 11).

Postscript

I was recently asked: "How many companies use your method?" My answer: "All of them, but most don't know it." I have yet to walk into a development organization and not find the techniques discussed in this text being used by developers and managers alike. It is as if the development organization has agreed to abide by dual management systems: one being formal, phase-based, long lead-time, finite, and largely fictitious, and the other being informal, unrecorded, incremental, continuous, and the one on which most decisions are based.

My colleague continued on the theme: "When I asked how many companies use your method?' you could have said, None. How many companies are using object-oriented analysis? None. How many companies are using UML or CMM? None. How many companies are using Extreme Programming? None. In my consulting, I have never seen two organizations employ any set of principles completely or exactly like any other company. All the development managers with whom I have worked say they employ some paradigm or technique, but they value some tenets and conveniently ignore others."

This is to be expected (refer back to Chapter 7). Most of the organizations using the techniques have adopted the parts that work with their environment. No organization can try to adopt a philosophy without reflection and adaptation — nor should they. The ideas presented in this text were defined and developed by talented women and men in dozens of companies with whom I have worked. It is far more constructive to think of the ideas in this book as a formalization of your own ideas, rather than management techniques someone else foisted upon you.

Thinking back to the original question — "How many companies use your method?" — I added one more nuance. The ideas expressed in this text are not mine. They represent the sum total of techniques I have seen work in development organizations around the world. I took note of how developers really worked (as opposed to what they reported in status reports). I assembled the techniques of effective project managers and combined them into the work you have just read.

Given that the source material for this book came from people like you, it makes sense to view the material as dynamic and evolving. There are countless questions development managers ask every day. Most are best addressed through dialogue with other managers and developers.

An active Web conference at http://luminguild.com/dynamic is available for exchange of questions, ideas, and observations. You are welcome to participate and enhance everyone's ability to manage software development.

Appendix A

Distorted Reality…

Or Why Phased Management Is Appealing

Planning development projects as a series of phases, where each phase is a type of work, leads to erroneous information and redundant effort.
Phased-based management leads to long lead-times.
Phased-based management leads to specialization of skills.
Phase-based management leads to distortion of the real objective.
Phased-based management leads to lost information assets.
Other than that, it is great!

What Is So Bad about the Waterfall?

I have searched for years to find a logical explanation for the popularity of phased methodologies. I have found none. Organizing a development effort into distinct phases with each phase characterized by the type of work being performed (e.g., in the design phase, we build designs; in the coding phase, we code) is convenient for planning and management. However, it creates a false sense of security. No one believes that after analysis we have a perfect understanding of the user's requirements or after the coding phase we no longer need to write code.

Why Long Lead-Times?

Phased management creates long lead-times because of the intrinsic feature of defining and approving all feasibility, requirements, design, code, and test information before the system is delivered. The phases tend to create long periods of time where the development teams and users have little interaction. This isolation fosters disparate expectations of the system, its features, and functions. When the system is finally delivered, the differences in expectations becomes clear, forcing the development effort to rework many aspects of the system. This correction cycle further delays the time when the system is providing value.

My opinions may have changed, but not the fact that I am right.

—Ashleigh Brilliant

Phased Management Leads to Specialization of Skills

Many managers find it useful to organize developers into specialists. They have an analysis group, an implementation group, and a database group. While some developers seem to like having kindred spirits in the next cubicle, this separation tends to produce a myopic view of the enterprise's needs. When all your colleagues are focused on analysis, the goal is to produce the best-quality functional specification. When all your colleagues are programmers, the goal is to produce the best-quality code. While this is laudable, the quality of the service provided to the enterprise is not limited to the quality of a single document. The priority should be the discovery, definition, implementation, and distribution of valuable systems. A beautiful specification that does not contribute to a beautiful design and test suite is not an asset. A set of modules that is difficult to test or implements an ambiguous specification is a liability. Separating skills into silos postpones the dialogue between developers and creates a contentious environment. It leads to a myopic view that allows developers to rationalize their role: "I did my part. What is wrong with you?" The development teams should be a collection of complementary skills organized by business need.

The early days of the software industry attracted the maverick developer. We encouraged and rewarded the rare, talented individual who could work mysteriously and deliver software solutions that appeared valuable. It has taken us decades to realize that developing complex information systems requires a diverse set of skills impossible for one person to master. No method, tool, or methodology can provide the lone developer with the expertise, critical review, multiple perspectives, or encouragement needed to keep an information system abreast of corporate needs. It is important for the analyst to participate in the integrated process of providing value. It is important that the tester work directly with the designer. It is less important for the analyst to hang out with other analysts and programmers or to consort with other programmers.

Phase-Based Management Leads to Distortion of the Real Objective

We are not working to create big solutions (big specifications, lots of code, etc.). We are working to add value.

Software systems supporting our businesses must respond quickly to allow adaptation to change that keeps us competitive. By defining the process as a set of preplanned phases, the goal shifts from adding value as opportunity presents itself to come in on time and under budget. Many system failures have been on time and within budget. They succeeded in meeting the numbers hysterically formulated to justify a major project effort. They failed because, as the project progressed, managers made decisions aimed at meeting the time and financial constraints at the expense of true user needs. When you hear someone say, "We can't change that requirement because it prevents us from meeting our deadline," it is a clear indicatation that the goals of the effort have shifted from meeting enterprise needs to satisfying the arbitrary constraints of "the project."

Incremental development (i.e., short, predictable cycles) always results in a better fit between our businesses and our systems without the disruption of long-cycle development projects. Software development is an ongoing, integral part of the enterprise — not a special case operation to be kept at the fringes of the business.

The measure of a software developer's work is how much value has been added to the enterprise's information asset. A project manager told me that her organization was very productive, working at 32 function points per person-month. I said that seemed very impressive but how do we know the function points are adding value? I would much rather know that three person-months of work made a hospital's patient data more valuable by adding access from procedure rooms, than that the effort bought us 49 function points. This measure is only visible in the perception of the system's users.

Phased-Based Management Leads to Lost Information Assets

Manage the software development effort as a knowledge-creating endeavor. The construction of a software system requires the use and creation of accurate information about the system's requirements, design decisions, implementation, and testing practices. The information developers create while building new features is critical to the ability to enhance and adapt those features to future requirements.

Partitioning a project into phases creates points in time where developers push to "finish." This usually means "record something that approximates what we have discovered — don't worry if it is accurate or complete — no one will be looking at it anyway." I have yet to find a developer working in a phase-based environment who is confident that the information recorded in the project repository is truly accurate. I have yet to see a maintenance programmer in a phased-based environment who goes to the requirements first when assigned an enhancement project. This tells us that the information in the repository is less than useless. It says that, in the opinion of the developer, using the information about the requirements will be more costly than recreating the knowledge from the existing implementation.

The reason the information in the repository is so inaccurate is that we do not recognize and reinforce its use in ongoing development efforts. Most managers with whom I have worked give lip service to the need for accurate archives, and yet they sabotage their own assets by rewarding developers who do whatever it takes to meet the deadline. It is no wonder that 70 to 80 percent of the software development budget goes toward maintenance.

This text defines a method that makes no distinction between enhancement and new development. The management of the software development effort is not divided between new features, which are directed by artificial project constraints, and enhancement, which tries to make up for the errors committed during new development projects.

Appendix B

Where to Begin…

Or Getting Started with Dynamic Management

Changing a management philosophy is a difficult and risky process. The best chance for success is to create situations where developers and managers have choices, but the easiest response is to do it the right way. Start by conducting a pilot effort with trusted colleagues in a quiet way.

The basic checklist (in chronological order) is:

1. Select a team.
2. Select initial projects.
3. Define elements and relationships.
4. Define a simple repository.
5. Report status in terms of features.
6. Track hours charged to the effort.

Do not announce or hype the experiment. It is best to just try the ideas you find promising without fanfare or exaggerated expectations.

It's easier to get forgiveness than permission.

Selecting Comrades

If you have been talking with other managers and developers about the ideas in this book, you probably have a good sense of who is interested and who is skeptical. You need to gather developers and colleagues you believe are open to innovation and willing to try making some small changes with the potential for big improvement. My experience suggests that developers relate to the ideas of dynamic management more readily than managers. You might find your best allies among your senior analysts and team leaders. Select from this cadre a team of four to seven developers. Choose developers with a diverse and complementary set of skills. The team should comprise individuals recognized for their skills in business analysis, database, programming, testing, and user documentation.

The message to your team is: "Our goal is providing consistent, effective, predictable solutions to the enterprise on a continuous basis (and improve ourselves in the process)."

Orchestrating Pilot Projects

Select two related projects that have been approved by your management. The best choices are projects that build upon a common set of data objects: a project that defines a portion of a database and another project that extends that database or provides additional functionality to the original version.

The projects should be relatively small. Projects you expect to require approximately five to ten person-months should work well.

Build the initial repository during the first project. Begin working on the objectives of the second without officially closing the first. The goal is to build momentum and get people thinking in a dynamic fashion before they realize there is something different.

Define Your Building Blocks

Meet with your development team and sketch out the elements and associations you feel are important to your collective understanding of the product. Refer to Chapter 3 for ideas. Define the first couple of tasks your team will address.

Define Your Repository

Discuss how the elements and tasks should be physically stored. A simple set of conventions and a shared text document might be enough. Your developers might be able to define an extension to your configuration management tool or CASE tool to support your effort. The physical form is not the issue. The important criterion is that everyone must have easy access and updated permission. The repository is the communication tool. Every member of the development team must agree to keep it up-to-date.

Report Feature

As the project progresses, issue regular updates to interested users and managers. The report should not report percent complete or which phase the team is in. The report should highlight the features that are in place and demonstrable (even if only demonstrable within a test environment). Your goal is to get other managers comfortable with the idea of reporting progress in the form of service that has been and will be provided to the enterprise.

Measure Productivity

At the end of each reporting cycle, apply the techniques discussed in Chapter 5. Allocate the hours for which you paid your developers to the elements that have been added to the repository. After a couple of cycles, discuss the results with the development team, paying attention to differences between the time periods. Do the numbers generated from the allocation correspond to the team's sense of its own productivity? If so, you have some evidence that this simple approach can provide you with valuable information without having to interpret developer written status reports.

What You Can Lose

Nothing is risk-free. If you experiment with different management techniques, you run the risk of upsetting colleagues and superiors comfortable and familiar with more traditional approaches. The reason you start with a set of small projects is to get through the learning curves without undue scrutiny. The small project helps gain allies and allows you to gather hard evidence of the technique's effectiveness.

If the first pilot effort is reasonably successful, expand it to clusters of tasks with larger scope involving more developers and different users. As people notice differences, explain what you are doing. Offer to show data you have gathered and to talk with people as to how they like the changes you have initiated. As more and more people notice changes, you can address any concerns one by one, and with more and more evidence to back up your position.

Soon, the organization's norms have changed to the point that no one has any reason to challenge your initiative. Dynamic management simply becomes the way things are done as opposed to another major reengineering/process improvement campaign foisted upon us from outside, consuming value resources, disrupting productive work, and suffering an embarrassing demise.

Appendix C

Capability Maturity Model and Dynamic Software Management

Most of the discussions I have heard concerning the implementation of the Capability Maturity Model^{SM} (CMM) assume that work is organized by phase. Yet there is no vocabulary in the CMM that suggests an organization must use a methodology that separates tasks into analysis tasks versus coding tasks. The CMM does not specify how a manager must measure productivity. Still, my conversations with other managers indicates a degree of discomfort in relating the CMM to dynamic software management. This appendix offers some thoughts on how the CMM and techniques presented in this text work together. I refer to chapters I believe relate to the elements. Where I do not have a comment about the direct relationship between the CMM and dynamic management, I leave the cell blank.

> It is better to sit in silence and appear ignorant, than to open your mouth and remove all doubt.
>
> **—Mark Twain**

LEVEL 1: REPEATABLE
Key Practice: Requirements Management

Capability Maturity Model	**Dynamic Management**
Goals	
Goal 1: System requirements allocated to software are controlled to establish a baseline for software engineering and management use.	System requirements allocated to software take the form of requirements-related elements "declared" in the repository.
Goal 2: Software plans, products, and activities are kept consistent with the system requirements allocated to software.	The premise of dynamic management is that development's goal is to move the repository from one consistent state to the next.
Commitment to Perform	
Commitment 1: The project follows a written organizational policy for managing the system requirements allocated to software.	Managers formalize the methods used to decide which elements are added to the repository.
Ability to Perform	
Ability 1: For each project, responsibility is established for analyzing the system requirements and allocating them to hardware, software, and other system components.	I recommend team leaders be delegated the responsibility and authority to make these decisions.
Ability 2: The allocated requirements are documented.	Documentation is the repository.
Ability 3: Adequate resources and funding are provided for managing the allocated requirements.	Funding of the development effort is tied directly to functionality. If the organization wants more functionality, they allocate more resources.
Ability 4: Members of the software engineering group and other software-related groups are trained to perform their requirements management activities.	
Activities Performed	
Activity 1: The software engineering group reviews the allocated requirements before they are incorporated into the software project.	Elements added to the repository are reviewed to move them to a verified state.
Activity 2: The software engineering group uses the allocated requirements as the basis for software plans, work products, and activities.	Elements related to requirements are usually the focus element of tasks.
Activity 3: Changes to the allocated requirements are reviewed and incorporated into the software project.	Elements changed in the repository are reviewed to move them to a verified state.

Measurement and Analysis	
Measurement 1: Measurements are made and used to determine the status of the activities for managing the allocated requirements.	Elements in the repository are the units of measure.
Verifying Implementation	
Verification 1: The activities for managing the allocated requirements are reviewed with senior management on a periodic basis.	Management reviews the status of the effort by monitoring changes in the repository.
Verification 2: The activities for managing the allocated requirements are reviewed with the project manager on both a periodic and event-driven basis.	
Verification 3: The software quality assurance (SQA) group reviews and/or audits the activities and work products for managing the allocated requirements and reports the results.	The SQA group monitors the repository.

Key Practice: Software Project Planning

Capability Maturity Model	**Dynamic Management**
Goals	
Goal 1: Software estimates are documented for use in planning and tracking the software project.	Most estimates are projected from information in the repository.
Goal 2: Software project activities and commitments are planned and documented.	Planning and documenting of project activities is integral to defining and assigning tasks.
Goal 3: Affected groups and individuals agree to their commitments related to the software project.	All parties must agree to base their work on a shared repository.
Commitment to Perform	
Commitment 1: A project software manager is designated to be responsible for negotiating commitments and developing the project's software development plan.	
Commitment 2: The project follows a written organizational policy for planning a software project.	
Ability to Perform	
Ability 1: A documented and approved statement of work exists for the software project.	This is problematic. The development plan typically evolves as the development teams work and users refine their requirements (Chapter 3)

Ability 2: Responsibilities for developing the software development plan are assigned.	
Ability 3: Adequate resources and funding are provided for planning the software project.	Dynamic management keeps a balance between resource and work (Chapter 9).
Ability 4: The software managers, software engineers, and other individuals involved in the software project planning are trained in the software estimating and planning procedures applicable to their areas of responsibility.	This is part of the development environment (Chapter 7).

Activities Performed

Activity 1: The software engineering group participates on the project proposal team.	
Activity 2: Software project planning is initiated in the early stages of, and in parallel with, the overall project planning.	Work on the development plan is continuous.
Activity 3: The software engineering group participates with other affected groups in the overall project planning throughout the project's life.	All groups should use the development plan as a focal point.
Activity 4: Software project commitments made to individuals and groups external to the organization are reviewed with senior management according to a documented procedure.	Task definitions are allocated to developers (internal or external).
Activity 5: A software life cycle with predefined stages of manageable size is identified or defined.	Stages are defined as related sets of tasks (Chapter 2).
Activity 6: The project's software development plan is developed according to a documented procedure.	
Activity 7: The plan for the software project is documented.	The plan is part of the repository.
Activity 8: Software work products that are needed to establish and maintain control of the software project are identified.	The software work products are the elements and the changes in the elements' state.
Activity 9: Estimates for the size of the software work products (or changes to the size of software work products) are derived according to a documented procedure.	Size estimates are generated from the element definitions and development plan.
Activity 10: Estimates for the software project's effort and costs are derived according to a documented procedure.	Cost estimates are generated from past task definitions.

Activity 11: Estimates for the project's critical computer resources are derived according to a documented procedure.	Estimates are generated from past task definitions.
Activity 12: The project's software schedule is derived according to a documented procedure.	Schedules are generated from past productivity rates.
Activity 13: The software risks associated with the cost, resource, schedule, and technical aspects of the project are identified, assessed, and documented.	Evaluating the development plan is used to assess risk.
Activity 14: Plans for the project's software engineering facilities and support tools are prepared.	The framework specifies the plan for support tools.
Activity 15: Software planning data is recorded.	The development plan is the record.
Measurement and Analysis	
Measurement 1: Measurements are made and used to determine the status of the software planning activities.	The states of tasks are recorded in the repository.
Verifying Implementation	
Verification 1: The activities for software project planning are reviewed with senior management on a periodic basis.	Regular review by senior management is an important part of the leadership (Chapter 10).
Verification 2: The activities for software project planning are reviewed with the project manager on both a periodic and event-driven basis.	
Verification 3: The software quality assurance group reviews and audits the activities and work products for software project planning and reports the results.	

Key Practice: Software Project Tracking and Oversight

Capability Maturity Model	**Dynamic Management**
Goals	
Goal 1: Actual results and performances are tracked against the software plans.	Monitoring the repository is the method of tracking progress (Chapter 5).
Goal 2: Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans.	
Goal 3: Changes to software commitments are agreed to by the affected groups and individuals.	The development plan evolves as the development teams work and users refine their requirements (Chapter 3).
Commitment to Perform	
Commitment 1: A project software manager is designated to be responsible for the project's software activities and results.	

Commitment 2: The project follows a written organizational policy for managing the software project.	
Ability to Perform	
Ability 1: A software development plan for the software project is documented and approved.	Refer to Chapters 3 and 10.
Ability 2: The project software manager explicitly assigns responsibility for software work products and activities.	Chapter 4 discusses the process of defining tasks and assigning them to teams.
Ability 3: Adequate resources and funding are provided for tracking the software project.	
Ability 4: The software managers are trained in managing the technical and personnel aspects of the software project.	Ensuring "fit" in the development environment includes the balance between training and management methods (Chapter 7).
Ability 5: First-line software managers receive orientation in the technical aspects of the software project.	
Activities Performed	
Activity 1: A documented software development plan is used for tracking the software activities and communicating status.	The development plan reflects the status of the repository. The status of the development effort is communicated by direct access to the repository and through regular interaction (Chapter 10).
Activity 2: The project's software development plan is revised according to a documented procedure.	
Activity 3: Software project commitments and changes to commitments made to individuals and groups external to the organization are reviewed with senior management according to a documented procedure.	
Activity 4: Approved changes to commitments that affect the software project are communicated to the members of the software engineering group and other software-related groups.	
Activity 5: The size of the software work products (or size of the changes to the software work products) are tracked, and corrective actions are taken as necessary.	The set of element definitions records the size metric for each element. Association definitions include average frequency. Together, we can project the size of proposed strands.
Activity 6: The project's software effort and costs are tracked, and corrective actions are taken as necessary.	Changes in cost and size are generated from the information in the repository.

Activity 7: The project's critical computer resources are tracked, and corrective actions are taken as necessary.	Resources for the perpetual project are defined in the framework (Chapter 6).
Activity 8: The project's software schedule is tracked, and corrective actions are taken as necessary.	Chapter 5 discusses projecting and monitoring schedules.
Activity 9: Software engineering technical activities are tracked, and corrective actions are taken as necessary.	
Activity 10: The software risks associated with cost, resource, schedule, and technical aspects of the project are tracked.	Risk is discussed in Chapter 9.
Activity 11: Actual measurement data and replanning data for the software project are recorded.	The repository contains information about the state of the product and the project.
Activity 12: The software engineering group conducts periodic internal reviews to track technical progress, plans, performance, and issues against the software development plan.	
Activity 13: Formal reviews to address the accomplishments and results of the software project are conducted at selected project milestones according to a documented procedure.	
Measurement and Analysis	
Measurement 1: Measurements are made and used to determine the status of the software tracking and oversight activities.	
Verifying Implementation	
Verification 1: The activities for software project tracking and oversight are reviewed with senior management on a periodic basis.	Regular review by senior management is an important part of the leadership (Chapter 10).
Verification 2: The activities for software project tracking and oversight are reviewed with the project manager on both a periodic and event-driven basis.	
Verification 3: The software quality assurance group reviews and/or audits the activities and work products for software project tracking and oversight and reports the results.	

Key Practice: Software Subcontract Management

Capability Maturity Model	Dynamic Management
Goals	
Goal 1: The prime contractor selects qualified software subcontractors.	Subcontractors should be treated as another team to which you allocate tasks.
Goal 2: The prime contractor and the software subcontractor agree to their commitments to each other.	Subcontractors agree to keeping the repository up-to-date and accurate.
Goal 3: The prime contractor and the software subcontractor maintain ongoing communications.	Ongoing communication is conducted via the repository (Chapter 4).
Goal 4: The prime contractor tracks the software subcontractor's actual results and performance against its commitments.	The development plan serves this purpose.
Commitment to Perform	
Commitment 1: The project follows a written organizational policy for managing the software subcontract.	
Commitment 2: A subcontract manager is designated to be responsible for establishing and managing the software subcontract.	
Ability to Perform	
Ability 1: Adequate resources and funding are provided for selecting the software subcontractor and managing the subcontract.	Subcontractors must have expertise in the enterprise's framework and agree to work with the enterprise's development environment (Chapters 6 and 7).
Ability 2: Software managers and other individuals who are involved in establishing and managing the software subcontract are trained to perform these activities.	
Ability 3: Software managers and other individuals who are involved in managing the software subcontract receive orientation in the technical aspects of the subcontract.	
Activities Performed	
Activity 1: The work to be subcontracted is defined and planned according to a documented procedure.	
Activity 2: The software subcontractor is selected, based on an evaluation of the subcontract bidders' ability to perform the work, according to a documented procedure.	

Activity 3: The contractual agreement between the prime contractor and the software subcontractor is used as the basis for managing the subcontract.	The contract must specify details of a shared repository.
Activity 4: A documented subcontractor's software development plan is reviewed and approved by the prime contractor.	The subcontractor's development plan must have the same content as the prime contractor's development plan.
Activity 5: A documented and approved subcontractor's software development plan is used for tracking the software activities and communicating status.	
Activity 6: Changes to the software subcontractor's statement of work, subcontract terms and conditions, and other commitments are resolved according to a documented procedure.	The shared repository forms the basis for defining tasks and monitoring changes.
Activity 7: The prime contractor's management conducts periodic status/coordination reviews with the software subcontractor's management.	The shared repository provides the means of monitoring development status (Chapter 5).
Activity 8: Periodic technical reviews and interchanges are held with the software subcontractor.	
Activity 9: Formal reviews to address the subcontractor's software engineering accomplishments and results are conducted at selected milestones according to a documented procedure.	
Activity 10: The prime contractor's software quality assurance group monitors the subcontractor's software quality assurance activities according to a documented procedure.	
Activity 11: The prime contractor's software configuration management group monitors the subcontractor's activities for software configuration management according to a documented procedure.	
Activity 12: The prime contractor conducts acceptance testing as part of the delivery of the subcontractor's software products according to a documented procedure.	
Activity 13: The software subcontractor's performance is evaluated on a periodic basis, and the evaluation is reviewed with the subcontractor.	

Measurement and Analysis	
Measurement 1: Measurements are made and used to determine the status of the activities for managing the software subcontract.	
Verifying Implementation	
Verification 1: The activities for managing the software subcontract are reviewed with senior management on a periodic basis.	Regular review by senior management is an important part of the leadership (Chapter 10).
Verification 2: The activities for managing the software subcontract are reviewed with the project manager on both a periodic and event-driven basis.	
Verification 3: The software quality assurance group reviews and audits the activities and work products for managing the software subcontract and reports the results.	

Key Practice: Software Quality Assurance Capability Maturity Model	Dynamic Management
Goals	
Goal 1: Software quality assurance activities are planned.	SQA activities focus on the changes in the state of elements in the repository.
Goal 2: Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively.	
Goal 3: Affected groups and individuals are informed of software quality assurance activities and results.	
Goal 4: Noncompliance issues that cannot be resolved within the software project are addressed by senior management.	
Commitment to Perform	
Commitment 1: The project follows a written organizational policy for implementing software quality assurance (SQA).	
Ability to Perform	
Ability 1: An group that is responsible for coordinating and implementing SQA for the project (i.e., the SQA group) exists.	
Ability 2: Adequate resources and funding are provided for performing the SQA activities.	

Ability 3: Members of the SQA group are trained to perform their SQA activities.	
Ability 4: The members of the software project receive orientation on the role, responsibilities, authority, and value of the SQA group.	
Activities Performed	
Activity 1: A SQA plan is prepared for the software project according to a documented procedure.	
Activity 2: The SQA group's activities are performed in accordance with the SQA plan.	
Activity 3: The SQA group participates in the preparation and review of the project's software development plan, standards, and procedures.	The SQA group has access to the repository and monitors its development.
Activity 4: The SQA group reviews the software engineering activities to verify compliance.	
Activity 5: The SQA group audits designated software work products to verify compliance.	Audits can occur at any time on any task and its associated strands.
Activity 6: The SQA group periodically reports the results of its activities to the software engineering group.	
Activity 7: Deviations identified in the software activities and software work products are documented and handled according to a documented procedure.	
Activity 8: The SQA group conducts periodic reviews of its activities and findings with the customer's SQA personnel, as appropriate.	
Measurement and Analysis	
Measurement 1: Measurements are made and used to determine the cost and schedule status of the SQA activities.	
Verifying Implementation	
Verification 1: The SQA activities are reviewed with senior management on a periodic basis.	
Verification 2: The SQA activities are reviewed with the project manager on both a periodic and event-driven basis.	
Verification 3: Experts independent of the SQA group periodically review the activities and software work products of the project's SQA group.	

Key Practice: Software Configuration Management

Capability Maturity Model	**Dynamic Management**
Goals	
Goal 1: Software configuration management activities are planned.	
Goal 2: Selected software work products are identified, controlled, and available.	Elements are the work products.
Goal 3: Changes to identified software work products are controlled.	Changes to elements move through a well-defined set of states (Declared through Verified).
Goal 4: Affected groups and individuals are informed of the status and content of software baselines.	Summary status information is based on the state of the repository.
Commitment to Perform	
Commitment 1: The project follows a written organizational policy for implementing software configuration management (SCM).	
Ability to Perform	
Ability 1: A board having the authority for managing the project's software baselines (i.e., a software configuration control board — SCCB) exists or is established.	I recommend that the development managers constitute the SCCB.
Ability 2: A group that is responsible for coordinating and implementing SCM for the project (i.e., the SCM group) exists.	I recommend that the team leaders constitute the SCM group.
Ability 3: Adequate resources and funding are provided for performing the SCM activities.	
Ability 4: Members of the SCM group are trained in the objectives, procedures, and methods for performing their SCM activities.	
Ability 5: Members of the software engineering group and other software-related groups are trained to perform their SCM activities.	
Activities Performed	
Activity 1: An SCM plan is prepared for each software project according to a documented procedure.	
Activity 2: A documented and approved SCM plan is used as the basis for performing the SCM activities.	
Activity 3: A configuration management (CM) library system is established as a repository for the software baselines	The CM library is an extension of the repository (or vice versa).

Activity 4: The software work products to be placed under configuration management are identified.	All defined elements are work products (Chapters 2 and 3).
Activity 5: Change requests and problem reports for all configuration items/units are initiated, recorded, reviewed, approved, and tracked according to documented procedures.	
Activity 6: Changes to baselines are controlled according to a documented procedure.	
Activity 7: Products from the software baseline library are created and their release is controlled according to a documented procedure.	
Activity 8: The status of configuration items/units is recorded according to a documented procedure.	
Activity 9: Standard reports documenting the SCM activities and the contents of the software baseline are developed and made available to affected groups and individuals.	These reports can be implemented as queries into the repository.
Activity 10: Software baseline audits are conducted according to a documented procedure.	
Measurement and Analysis	
Measurement 1: Measurements are made and used to determine the status of SCM activities.	
Verifying Implementation	
Verification 1: The SCM activities are reviewed with senior management on a periodic basis.	
Verification 2: The SCM activities are reviewed with the project manager on both a periodic and event-driven basis.	
Verification 3: The SCM group periodically audits software baselines to verify that they conform to the documentation that defines them.	
Verification 4: The software quality assurance group reviews and audits the activities and work products for SCM and reports the results.	

LEVEL 3: DEFINED

Key Practice Area: Organization Process Focus

Capability Maturity Model	Dynamic Management
Goals	
Goal 1: Software process development and improvement activities are coordinated across the organization.	Improvements consist of enhancements to the element definitions, refinement of the framework, and improvement in the development environment.
Goal 2: The strengths and weaknesses of the software processes used are identified relative to a process standard.	Refer to Chapter 7.
Goal 3: Organization-level process development and improvement activities are planned.	

Key Practice Area: Organization Process Definition

Capability Maturity Model	Dynamic Management
Goals	
Goal 1: A standard software process for the organization is developed and maintained.	Refer to Chapter 7.
Goal 2: Information related to the use of the organization's standard software process by the software projects is collected, reviewed, and made available.	

Key Practice Area: Training Program

Capability Maturity Model	Dynamic Management
Goals	
Goal 1: Training activities are planned.	Training is part of the environment evaluated in Chapter 7.
Goal 2: Training for developing the skills and knowledge needed to perform software management and technical roles is provided.	
Goal 3: Individuals in the software engineering group and software-related groups receive the training necessary to perform their roles.	

Key Practice Area: Integrated Software Management

Capability Maturity Model	Dynamic Management
Goals	
Goal 1: The project's defined software process is a tailored version of the organization's standard software process.	The organization's process is defining work by feature to be added, or enhanced, in the repository. The individual develop efforts are characterized by the selection and sequencing of specific features.

Goal 2: The project is planned and managed according to the project's defined software process.	There is no difference between maintenance and new development. Development efforts are managed by monitoring the state of the repository.

Key Practice Area: Software Product Engineering

Capability Maturity Model	**Dynamic Management**
Goals	
Goal 1: The software engineering tasks are defined, integrated, and consistently performed to produce the software.	
Goal 2: Software work products are kept consistent with each other.	Insuring consistency is the central and most basic theme of dynamic management.

Key Practice Area: Intergroup Coordination

Capability Maturity Model	**Dynamic Management**
Goals	
Goal 1: The customer's requirements are agreed to by all affected groups.	All affected groups use the repository to coordinate their activities (Chapter 4).
Goal 2: The commitments between the engineering groups are agreed to by the affected groups.	
Goal 3: The engineering groups identify, track, and resolve intergroup issues.	

Key Practice Area: Peer Reviews

Capability Maturity Model	**Dynamic Management**
Goals	
Goal 1: Peer review activities are planned.	Review and error detection/correction cycles should be integral to the development team.
Goal 2: Defects in the software work products are identified and removed.	

LEVEL 4: MANAGED

Key Practice Area: Quantitative Process Management

Capability Maturity Model	**Dynamic Management**
Goals	
Goal 1: The quantitative process management activities are planned.	The development manager develops queries against the state of the repository to measure development activity.
Goal 2: The process performance of the project's defined software process is controlled quantitatively.	
Goal 3: The process capability of the organization's standard software process is known in quantitative terms.	

Key Practice Area: Software Quality Management

Capability Maturity Model	Dynamic Management
Goals	
Goal 1: The project's software quality management activities are planned.	The development manager develops queries against the state of the repository to measure quality.
Goal 2: Measurable goals for software product quality and their priorities are defined.	
Goal 3: Actual progress toward achieving the quality goals for the software products is quantified and managed.	

LEVEL 5: OPTIMIZING

Key Practice Area: Defect Prevention

Capability Maturity Model	Dynamic Management
Goals	
Goal 1: Defect prevention activities are planned.	Defect prevention is best handled through effective team organization and reward.
Goal 2: Common causes of defects are sought out and identified.	It is the role of the development manager to monitor the changes being made to the elements in the repository. Frequent changes to the same element and/or by the same team signals the need for changes in the development environment.
Goal 3: Common causes of defects are prioritized and systematically eliminated.	

Key Practice Area: Technology Change Management

Capability Maturity Model	Dynamic Management
Goals	
Goal 1: Incorporation of technology changes is planned.	This is handled through the dynamic maintenance of the development framework (Chapter 6).
Goal 2: New technologies are evaluated to determine their effect on quality and productivity.	
Goal 3: Appropriate new technologies are transferred into normal practice across the organization.	

Key Practice Area: Process Change Management

Capability Maturity Model	Dynamic Management
Goals	
Goal 1: Continuous process improvement is planned.	Chapter 7 describes a plan for continuous improvement to the development environment.
Goal 2: Participation in the organization's software process improvement activities is organizationwide.	This requires a significant degree of leadership on the part of development managers (Chapter 10).
Goal 3: The organization's standard software process and the projects' defined software processes are improved continuously.	

Capability Maturity Model for Software, Version 1.1. Copyrighted by Carnegie Mellon University, 1996.

Appendix D

Dynamic Management Information Model

This appendix contains a specification for the information model used to manage dynamic software development (see Exhibit 1). You can use it to create a database and application supporting the dynamic management model defined in this book. The model also serves as a blueprint for identifying where this knowledge already exists within your current project repository (i.e., within your configuration management systems, document repository, or development environment tools).

Information Model Definitions

Conventions:

Symbol	*Read As*
=	consists of
+	along with
[x | y | z]	either x or y or z — one and only one
{ w }	some number of w's — zero or more

Database table names are in parenthesis.

Italics denotes a primary key.

The data attribute called "object id" is a surrogate key assigned when an object is created with a value unique across the entire model.

The term "ref" indicates a logical associate (usually implemented as a foreign key).

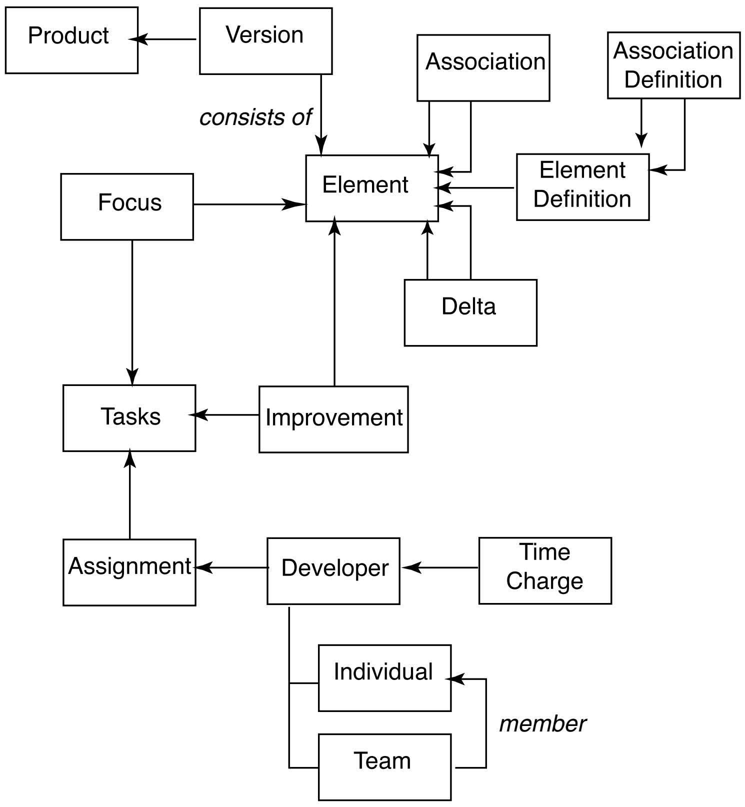


Exhibit 1 Information Model

Element Definition (ElementDefs) =

element name + element's purpose + weight unit + *object id*
/* Description: a description of a type of element recorded in the repository */

Element (Elements) =

name + description + version + weight + state + { location } + element definition ref + *object id*
/* Description: a unit of information about the product */
weight = /* size or complexity metric */
state = [Declared | Uncertain | Verified | Change-Pending | Obsolete]
location = /* path and file name of document containing the element's definition */
(normalized) location (ElementLocations) = element ref + location
/* there may be many documents defining an Element */

Association Definition (RelationDefs) =
element1 ref + meaning from1 + average frequency1 + element2 ref + meaning from2 + average frequency2 + *object id*
/* Description: A description of a type of association between two elements recorded in the repository */
average frequency = /* used in estimations — approximate number of associations of this type in which the Element will participate */

Association (Associations) =
element1 ref + *element2 ref* + *association definition ref*
/* Description: An important relationship between two elements. Each element participating in the association is best understood in the context of the other. */

Product (Products) =
name + description + *object id*
/* Description: A set of functionality and data supporting user's needs */

Version (Versions) =
product ref + version number
+ baseline name + *object id*
/* Description: A version of the product */

Consists of (Consists) =
version ref + *element ref*
/* Description: A set of functionality and data supporting user's needs */
/* references only "defining" Elements */

Delta (Deltas) =
old element ref + *new element ref*
/* Description: Trail of identifiable versions of the element */

Task (Tasks) =
created date/time + description + element count + *object id*
/* Description: A definition of work to be performed */
/* element count = number of elements [estimated | actual] potentially effected by the proposed task */

Focus (Focus) =
task ref + *element ref*
/* Description: Identifies the element on which the developer should focus his/her efforts while working on a task */

Improvement (Improvements) =
task ref + *element ref* + *date/time* + transition type
/* Description: Reflects a change to a element's state made while working on a task */
Transition Type = [create | change | verify | associate]

Developer =
call name + *object id*
/* Description: a person or team responsible for performing tasks */
Individual = Developer + family name + title + cost per unit
Team = Developer + date created
Member = team ref + developer ref + date assigned

Time Charges =
developer ref + *from date* + *to date* + work units
/* Description: Days or hours of a developer's time charged to the project */

Assignment =
developer ref + *task ref* + date assigned
/* Description: A directive instructing a developer to work on a task */

Appendix E

Glossary

This glossary lists the author's interpretations and definitions of key words and phases used in the text.

Accrued Work: Elements identified as being necessary but, as yet, unfinished.
Architecture: A set of collective decisions the organization makes governing the shape of the systems it creates (e.g., network hardware, operating systems, design techniques, interface standards, end-user equipment).
Asset Development: The concept of funding the development effort to develop and enhance information assets rather than funding to complete projects.
Association: A meaningful relationship between elements; the information in the repository that relates one element with another.
Association Definition: A declaration of a remembered relationship between two elements, specifying each participating element, the meaning of the association (from each element's perspective), and the average frequency of participation.
Cardinality: The allowable number of associated elements (*see* Frequency).
CASE Tool: Computer Aided Software Engineering Tool.
Change Pending Element: An element the development team knows must be altered, but the work has not been completed (*see* Element State).
Chunk: Information stored and retrieved as a single entity.
Class: An abstract definition of the attributes and methods of an Object.
Collaborate Channel: Communication between managers and supervisors within the development organization.
Communication Channel: A means/path of communication (*see* Collaborate Channel, Delegate Channel, Service Channel).
Component Inventory: A list of all components in the development environment; a list of all instances of standards, tools, measurements, methods, rewards, reviews, and training.

Decision Mapping: The process of identifying a need and deciding on the corresponding element of the framework. For example, a class defined in a design needs to be implemented; therefore, we write a Java class. A user describes a new feature; therefore, we add a use case to our repository.

Declared Element: An element the development team knows must be created, but has not been built (*see* Element State).

Decision Council: A committee or body charged with selecting and prioritizing development proposals.

Delegate Channel: Communication between a manager and his or her subordinates.

Design: Some statement of how a requirement will be implemented.

Developer: An individual or team assigned a particular task.

Development Campaign: An ongoing effort to support the development of some organizational domain.

Development Manager: Mid-level manager responsible for more than one development campaign.

Development Plan: List of active and planned tasks specifying the focus element(s), element count task description, task priority and sequence, and the assigned developer/team.

Development Repository: The set of all information (regardless of physical form or location) that describes all the products built and maintained by a development organization.

Dissonance List: A list of all development environment components conflicting in such a way as to hinder the development effort.

Documentation: The content of the repository; a record of relevant project and product information.

Domain: An application area usually associated with a cohesive set of business practices and a common set of users.

Element: An identifiable unit of the development repository; a chunk of information of a known type.

Element Count: A projected number of elements in a strand.

Element Size: Determined by: (1) the type of element; (2) the complexity of the element.

Element State: The current status of an item in the repository [declared | change pending | uncertain | verified].

Element Type: A classification of an element (e.g., requirement, design, object, test case).

Element Worksheet: A list of work products describing potential element descriptions, where the elements are physically located and a list of associated elements, used to define the contents of the repository.

Enhanced Value: The goal of communication: enhancing the value of information shared on a communication channel.

Entity: A thing of interest to a business organization (a.k.a. Object), usually representing information to be stored in a database.

Environment: A set of standards, tools, measurements, methods, rewards, reviews, and training used in the development process.

Event: Some occurrence causing the system to respond, usually some transaction or a point in time.

Focus: An element around which a task's objectives are framed.

Framework: A model of how software is implemented; a statement of how elements of existing technology are employed to provide service to the enterprise; a description of a generic implementation.

Frequency: Average number of associated elements existing in a repository (*see* Cardinality).

Good Code: Code that is consistent with the design and performs/delivers functionality described in the requirement.

Good Design: Design that describes the implementation of a requirement and an implementation can be produced from it.

Good Requirement: A statement of user expectations that is unambiguous and specific.

Good Test Case: The design of an experiment that has a high probability of detecting error, consisting of a description of an initial condition, a stimulus, and an expected result.

Improvement Log: A list of changes of state of the elements in a task's strand; represents work performed.

Information Structure: The scheme or organizing principles of a repository.

Legacy System: Applications still performing useful work but implemented with technology no longer used for current development.

Maintenance: The real work of software development; moving an existing product from one verified state to another verified state.

Measurement: An observable, objective scale (quantitative or qualitative) used to assess quality, volume, or duration.

Methodology: A set of task definitions.

Module: A portion of a design implemented using a computer language.

New Development: A classification of development work assuming no previous work needs to be considered; work is not based on any existing product (*see* Maintenance).

Object: A thing of interest to a business organization (a.k.a. Entity) usually representing a collection of data manipulated by code.

On-line Conference/Discussion: A networked facility allowing threaded discussion and shared documents.

Perpetual Effort: The concept that the development effort is continuous and fluid rather than project based.

Phase: A period of time planned for a particular type of work (e.g., analysis phase, testing phase).

Plan: A set of tasks ordered in a time sequence.

Planning Dialogue: Interaction among managers, developers, and users aimed at defining and prioritizing proposed tasks.

Political Boundary: Administrative or organizational edges marking authority, influence, and responsibility: division, department, committee.

Politics: The process of influencing those over whom you have no direct authority.

Process Improvement: Act of monitoring development practices and actively seeking ways to increase value: reduce error, increase productivity, enhance the developer's environment, improve staff development.

Product: A named set of elements perceived by the user as a single utility.

Product Size: Determined by: (1) the number of elements and associations in the repository; (2) the complexity of the elements.

Product State: The user's perception of the value of a software product (liability/asset).

Productivity: Number of changes to the repository per unit of work; weighted sum of element state changes divided by time charged to the tasks.

Release: A set of functionality made available to the user.

Repository: *see* Development Repository.

Requirement: A statement of the user's expectation. Requirements can be narrative, use case models, augmented information models.

Review: Any means of assuring the quality of a product and the adherence to standard or plan.

Reward: Any means of recognizing and encouraging desirable behavior or results.

Risk: The source and probability of failure.

Risk — Human Effort: Risk increases as the number of people involved increases.

Risk — Requirements: Risk increases as the use of untried technology increases.

Risk — Technology: Risk increases as increasingly employee untried technology.

Risk — Time Horizon: Risk increases as the time horizon expands.

Service Channel: Communication between development managers and user managers.

Shared Element: An element that is part of two active tasks; requires coordination between developers.

Size: *See* task size, product size, element size.

Skill Set: Knowledge and expertise needed to create and maintain an element type and its associated elements.

Standard: A statement of the observable attributes of an acceptable product.

Strand: The collection of elements constituting the subject of a task's work; a collection of elements associated with a focus element.

Strategic: Adjective referring to decisions and actions concerning the enterprise's future direction.

Tactical: Adjective referring to decisions and actions concerning the effectiveness of the enterprise today.

Task: A description of the value to be added to the repository, the strand of elements, and current development assignment.

Task Size: Determined by three factors: (1) the number of elements and associations in the strand; (2) the type of work to be done; (3) the complexity of the elements.

Task State: A summary of the state of elements with the strand associated with the task.

Team: Two or more developers with complementary skill sets sufficient to work on a task.

Test Case: An initial state, input, or stimulus and expected result.

Test Suite: A set of test cases organized in a series.

Tools and Techniques: Software and procedures used to aid and direct the development work.

Traceability: The ability to find related elements of information.

Training/Education: A means of acquiring working knowledge or useful information.

Uncertain Element: An element that is functional but has not been reviewed (*see* Element State).

Uncertainty: The ratio between number of verified elements and the total number of elements in the repository.

Verified Element: An element that is functional and reviewed (*see* Element State).

Version: An identified set of elements that are internally consistent and verified.

War Room: A shared space (usually a conference room) dedicated to a development team.

Weight: A predictive measure of relative size or complexity of an element or task.

Weight Unit: A unit of measure (e.g., paragraph count, kilobyte, edge).

Appendix F

Reading List

There is nothing new under the sun. All thought is an extension and rearrangement of previous thought. Here is a list of references I find useful — not always because the authors agree with me, but because they provide valuable insight and food for constructive thought.

1. Alexander, Christopher, *A Pattern Language: Towns, Buildings, Construction,* Oxford University Press, 1977.
2. Alexander, Christopher, *Notes of the Synthesis of Form,* Harvard University Press, 1970.
3. Beck, Kent and Fowler, Martin, *Planning Extreme Programming,* Addison-Wesley, 2001.
4. Beck, Kent, *Extreme Programming Explained,* Addison-Wesley, 2000.
5. Block, Robert, *The Politics of Projects,* Yourdon Press, 1983.
6. Boehm, Barry W., et al., *Software Cost Estimation with COCOMO II,* Prentice-Hall, 2000.
7. Capability Maturity Model for Software, Version 1.1, Carnegie Mellon University, 1996.
8. Cockburn, Alistair, *Agile Software Development,* Addison-Wesley, 2001.
9. DeMarco, Tom and Lister, Tim, *Peopleware: Productive Projects and Teams,* 2nd ed, Dorset House, 1999.
10. Frame, J. Davidson, *The New Project Management: Tools for an Age of Rapid Change, Corporate Reengineering, and Other Business Realities,* Jossey-Bass, 1994.
11. Highsmith, James, *Adaptive Software Development — A Collaborative Approach to Managing Complex Systems,* Dorset House, 1999.
12. Hoch, Detlev J., et al., *Secrets of Software Success — Management Insights from 100 Software Firms around the World,* Harvard Business School Press, 1999.
13. Humphry, Watts, *Introduction to the Personal Software Process,* Addison-Wesley, 1996.
14. Kerzner, Harold, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling,* 7th ed., John Wiley & Sons, 2000.

15. Kruchten, Philippe, *The Rational Unified Process, An Introduction,* 2nd ed., Addison-Wesley, 2000.
16. Lewin, Roger and Regine, Birute, *Weaving Complexity and Business: Engaging the Soul at Work,* Simon & Schuster, 1999.
17. McConnell, Steve C., *Rapid Development: Taming Wild Software Schedules,* Microsoft Press, 1996.
18. Mercadal, Dennis, *Dictionary of Artificial Intelligence,* Van Nostrand Reinhold, 1990.
19. Nosek, John T., *The Case for Collaborative Programming,* Communications of ACM, 41(3), 1998.
20. Page-Jones, Meiler, *Practical Project Management,* Dorset House, 1985.
21. Peter, Laurence J. and Hull, Raymond, *The Peter Principle: Why Things Always Go Wrong,* William Morrow & Company, 1969.
22. Pinchot, Gifford and Elizabeth, *The Intelligent Organization,* Berrett-Koehler Publishers, Inc., 1994.
23. Stewart, Thomas, *Intellectual Capital — The New Wealth of Organizations,* Doubleday, 1997.
24. Tinnirello, Paul C., Ed, *Handbook of Systems Development,* Auerbach, 1999.
25. Tinnirello, Paul C., Ed, *Project Management,* Auerbach, 2000.
26. Tufte, Edward, *Envisioning Information,* Graphic Press, 1990.
27. Umbaugh, Robert E., Ed, *Handbook of IS Management,* Auerbach, 1999.
28. Williams, Laurie, et al., *Strengthening the Case for Pair Programming, IEEE Software,* 17(4), 2000.
29. Wurman, Richard Saul, Ed., *Information Architects,* Graphics Press Corp., 1996.
39. Yourdon, Ed, *Managing High-Intensity Internet Projects,* Prentice-Hall, 2002.

Web Sites

1. *Dynamic System Development Method* — rapid development method emphasizing team empowerment and frequent product delivery. http://www.dsdm.org/index.asp
2. *Coad's Feature Driven Development (FDD)* — a model-driven, short-iteration process. http://www.togethercommunity.com/coad-letter/Coad-Letter-0070.html
3. *eXtreme Programming (XP)* — a deliberate and disciplined methodology stressing teamwork, feedback, and simplicity. http://www.extremeprogramming.org/
4. *Microsoft Solutions Framework (MSF)* — model-based development methodology focusing on distributed Internet computing. http://www.microsoft.com/business/services/mcsmsf.asp
5. *Object-oriented Process, Environment, and Notation (OPEN)* — OPEN Consortium's object-oriented methodology/process. www.open.org.au
6. *Rational Unified Process (RUP)* — a Web-enabled, team-based software engineering process. http://www.rational.com/products/rup/index.jsp
7. *Scrum* — a lightweight, agile process emphasizing a team-based approach to iteratively, incrementally develop software. Scrum refers to the mechanism used in rugby for getting an out-of-play ball back into play. http://www.controlchaos.com/
8. *Unified Modeling Language (UML)* — http://www.rational.com/uml/index.jsp

Appendix G

DSM Case Study

This appendix presents the case study illustration in its entirety and in chronological order. Excerpts are included at the end of each chapter.

Overview

DSM International provides a wide variety of services to its customers. The firm is headquartered on the West Coast with marketing and development organizations in three cities and two agent offices overseas.

Organization

DSM International provides a wide variety of services to its customers, including advisory consulting, training, industry research, and related software products. The firm grew from a three-person partnership serving a small group of local clients in San Francisco to a major international player in the field. Thirty years later, its headquarters is still in San Francisco but it has marketing and customer support offices in six U.S. cities and four agent offices overseas. Amazingly, all three of the original partners are still with the firm. One of the partners was born in India and now heads up the New Delhi operation. Another had family in Ireland and moved there six years ago to set up offices to serve DSM European clients.

The company's growth has not been spectacular, but it has been steady. In many ways, DSM's reputation and influence has grown faster than its revenue and profitability. Officers of the company feel good about this but would like to turn some of that reputation into real cash. Ten years ago, there was a big push to grow and expand the consulting side of the business. The company tripled the number of employees and contractors over a period of four months. They quickly discovered that increased revenue did not automatically translate

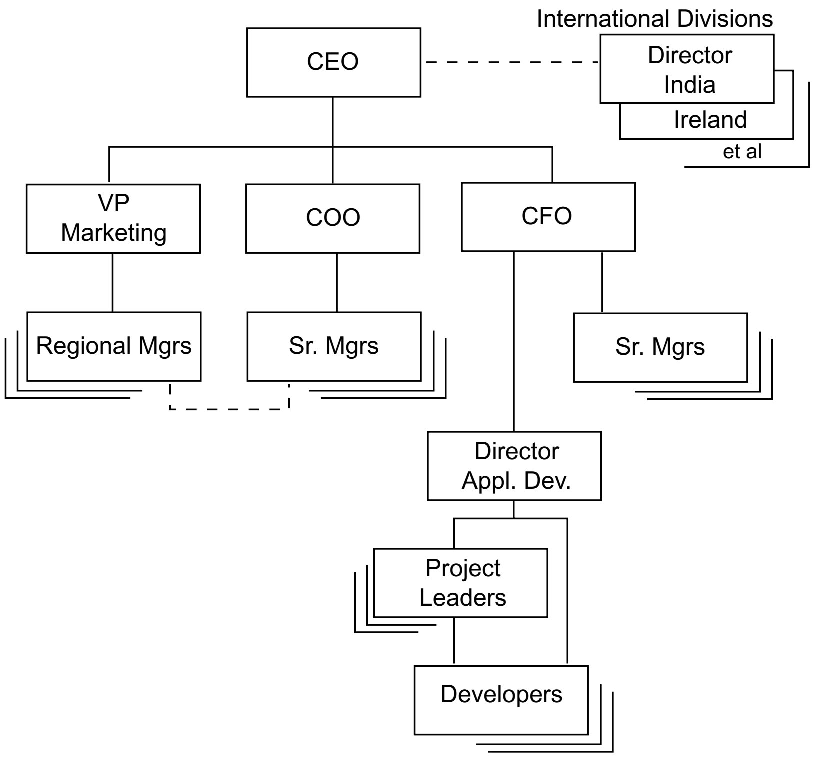


Exhibit 1 Old Organization Chart

into increased profit. The expansion was followed by a difficult period of very tight cash flow troubles. The employees and officers pulled through, but the period illustrated how the company learns — a never-ending cycle of good intentions and hard knocks. Through it all the culture is still open to change and internal improvement. While the industry generally views DSM as an exemplar, internally everyone from the CEO to the janitor knows there is always room for improvement. Determining which decisions will yield improvements and which are doomed to fail is the open question.

The current CEO is the third of the original founders of DSM. From the very beginning, he and his partners divided up the work of the firm into three domains: Marketing, Operations, and Finance. The basic structure of the company has remained the same even as the number of people in each of the areas has grown (see Exhibit 1).

Reporting to the VP of Marketing are regional managers. Each manager is responsible for client development in his or her geographic region. Reporting to the COO are senior managers responsible for developing and delivering services, many of which are specific to certain industries. These people and their subordinates work with clients, providing advisory consulting, delivering training seminars, and preparing industry-sector research reports. The managers under the COO are often assigned to assist in client development within given regions. The marketing and operations areas of DSM function in a loose, quasi-formal matrix organization. There is a long-standing rivalry between the two groups, with the managers in operations thinking of the regional managers as overblown salespeople overhyping the company's capabilities. The people in marketing refer to operations as a bunch of prima donnas earning their salary off the hard work of the marketing group. Both groups rely on the Applications Development group for internal systems and product software licensed to clients.

The Applications Development staff of the company currently has approximately 50 developers. They have outgrown the seventh floor; some developers are now located on different floors. The department is responsible for both in-house systems and customer products. DSM never found the need to separate in-house information technology from product development. The systems built and licensed to customers are similar to those originally built to support DSM's administrative and development functions. The development staff has successfully used DSM as a beta site for new development and major enhancements.

The CFO assumed the responsibility of buying computing equipment and maintaining the systems from the beginning. As the firm grew comfortable with the systems, it began relying less on their vendors for packaged solutions. Some of the accounting staff began writing their own applications. As these employees spent more time on software and systems and less time on accounting, the CFO modified her organization to coordinate the development effort. Several years later, there was a position defined for Director of Applications Development.

Most of the Applications Development department is located on the seventh floor of the San Francisco headquarters building. It is a typical configuration with developers housed in cubicles in the center of the floor and manager's offices around the perimeter. There is also a kitchen/lunch room, a testing lab, and a conference room used for everything from project status reviews to birthday parties.

Environment

The DMS development group prides itself on being able to deliver its products on a variety of platforms. The original systems were built on old Digital Equipment Corp. machines. The first product delivered to customers was on a UNIX platform. As the company grew and its client base expanded, the marketing group pushed to deliver Mac-based versions to clients in the entertainment and design sectors. DMS saw the need for Windows-based systems and delivered a Win 3.1 version soon after its release. The complement of developments reflects the diverse client base. As it happened, all the UNIX developers situated themselves together (see Exhibit 2). The Mac developers took a set of offices so they could easily work together. The growing number of Windows developers also flocked together. It was not by deliberate design, but when it was time to move to the seventh floor, the groups staked out their cubicles to establish territory. Now, the center isle is known as "Windows row." The outside rows of cubicles are referred to as "Mac row" and "UNIX row."

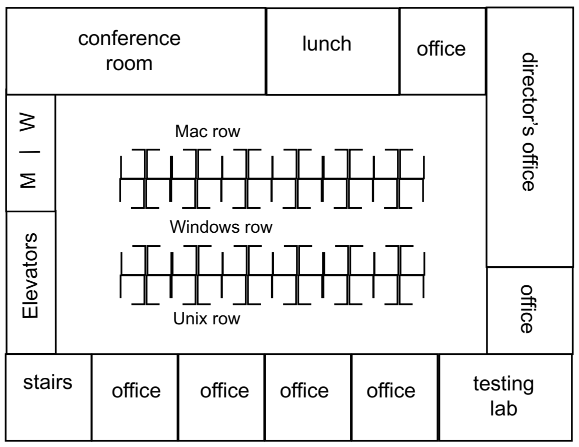


Exhibit 2 DMS Application Development Office Layout

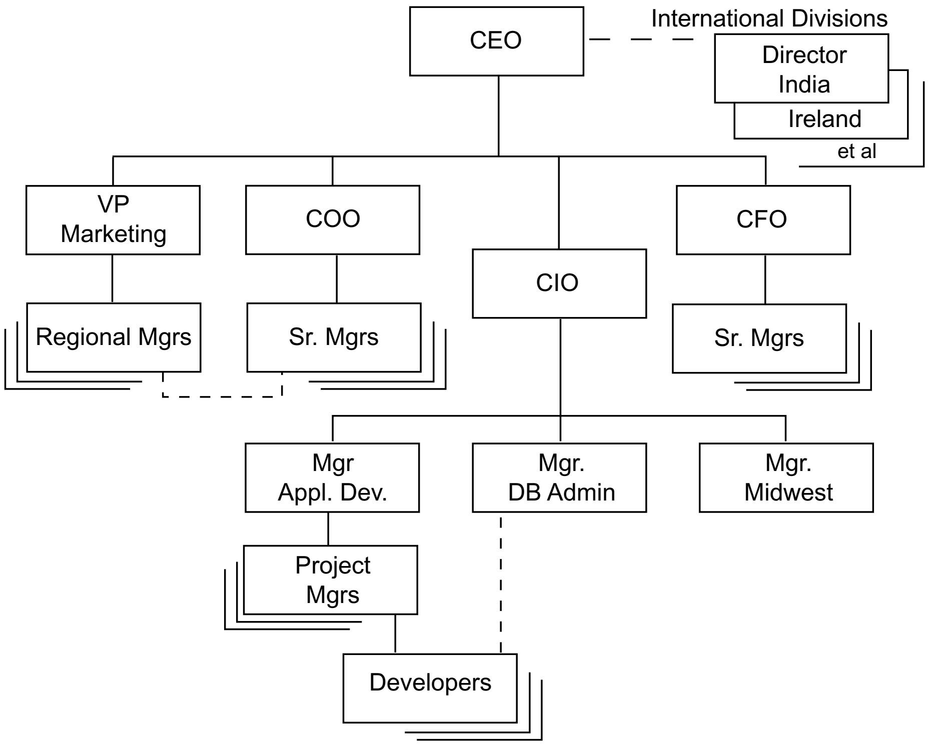


Exhibit 3 Recent Organization Chart

The organization not only works with multiple platforms, but also maintains and enhances systems from many generations. Many of the internal systems were developed in third-generation languages. These systems are well respected and valuable to DSM divisions (see Exhibit 3). If DSM had had a high turnover rate of developers, the issue of converting systems may have

Exhibit 4 Frustration from the VP of Marketing

To:	CEO
From:	VP of Marketing
Subject:	Support from Applications Development

We need more support from applications development. We have submitted many requests for new and updated systems. While it appears the people within applications development have the interest and the expertise, they clearly do not have the time. We are told that they are busy with updates to the financial applications and that our requests will be addressed when resources are available. In my conversations with the CFO, she claims she has no more money to hire more developers and that her applications will continue to take precedence.

This is untenable. We would prefer working with the internal development group but there are third-party vendors with software systems that will fill most of our needs. I need to know if I should continue to investigate outside sources for system support or if there will be development resources available that I can count on.

cc:

CFO

COO

Director of Applications Development

International Directors

taken a higher priority. But there are many developers and managers who have been with the company for a long time, providing the knowledge and experience to keep the system evolving and functional.

Time: Twelve Years Ago

DSM had more requests for system support than it could handle. Marketing wanted customer support for the regional managers, Operations wanted systems to aid the consultants, and the Financial people had a long list of enhancements to internal accounting systems. With the Director of Applications Development reporting to the CFO, the financial applications were getting preferred attention.

The VP of Marketing sent a memo to the CEO expressing his frustration, as shown in the memo in Exhibit 4. He also copied the memo to the other executives in DMS because he knew they were feeling the same frustration and wanted to ensure that the debate was enlarged to include all parts of the DMS enterprise.

The COO and directors of the international divisions replied with similar requests. They all said they would prefer working in-house, but that the level of service was not adequate to support the company's growing needs. The CFO requested more money to hire more developers and upgrade the development tools. The CEO was a little confused by all the discussion but relied upon his officers, agreeing to amend the budget and increase the investment in the systems development area.

The CEO's confusion came from the fact that everyone was talking about applications development as if the systems were capital assets — like they were requesting more plant and equipment, but the company was accounting for the development area like a clerical function. It was a cost center. Usually, everyone tried to minimize the money consumed by cost centers; but here, there was almost unanimous agreement that the company should "invest" in the division. The CEO wondered how he would know if the "investment" yielded a positive return.

The CEO had several conversations with the Director of Applications Development. He asked the Director how he felt about the current state of affairs. The Director reported that it was awkward. He wanted to be able to work with the other divisions on marketing and customer support, but until recently was only able to focus on the financial systems. They talked about the possibility of moving the development function into the marketing area or have several development groups — one for marketing, one for finance, one for operations. They agreed that it would be difficult to coordinate and that it seemed like a duplication of effort. Both men thought about the idea of Applications Development reporting directly to the CEO, but neither actually verbalized it. The CEO was uncomfortable with the technology and, in general, it seemed like an odd notion.

The CEO explained to the Director that he was increasing the funding for applications development and that he was sure the CFO would support the development of nonfinancial systems requested by other divisions of the company.

In the early years, as DSM development efforts grew in size and complexity, managers and developers alike recognized the need for more structure in the way software was built. Industry trade journals were filled with articles debating various methodologies and their characteristics: waterfall, spiral, model-driven, and process-oriented methods, all focusing on "process." Consulting firms preached analogies to other engineering and manufacturing practices. The managers and developers at DSM adopted the analogy of building a house with the need for careful plans and blueprints. Construction began only after careful planning; a firm foundation is laid before walls and roof are added. Enhancements identified after construction begins are postponed and handled by workers under a different contract.

DSM bought into the current thinking of the day and wanted a standard methodology for all its new development projects. Maintenance was to be treated as a "mini" new-development project. DSM contracted with a consulting firm to help define the methodology to be used in all of its development. The consultants worked with the managers and senior analysts to try to customize the consulting firm's template methodology as best they could. At the end of a difficult six months, the methodology was delivered in 17 beautiful binders called the DSM Development Methodology (DSM-DM). DSM paid the consultants a small phenomenal fee and set about the task of using the new procedures they had defined for themselves.

Before DSM adopted the DSM-DM, work was done at CMM level 1. That is not to say that the work was bad. In fact, the development group had a good reputation throughout DSM for being fairly responsive and for producing systems that worked and were reasonably usable. When the development organization was smaller, it was workable to have each developer keep track of his or her own documentation. But as the group grew in size and the systems they were building grew in complexity, documentation became unworkable. Each developer's unique idiosyncrasies made it awkward and time-consuming to share information and build upon each other's work.

There were mixed feelings about working with the outside consulting firm in writing the DSM-DM, but there was general consensus that it was a positive step. After all, everyone seemed to agree that a standardized set of phases in the development process was a good thing. Each phase would have a well-defined set of pre-conditions and expected results. The process would move smoothly from beginning to end. And everyone seemed to agree that there were (or should be) a definite beginning and a definite end to each development effort.

The development group of DSM was also very interested in the CMM and most thought that the idea of such a framework was valuable. The first articles and books published that explored the idea of a set of observable attributes of a competent organization caught the imagination of many of DSM's management. DSM never considered formal appraisal, but did invest a great deal of time and energy into modifying their methodology (DSM-DM) to reflect the Carnegie Mellon University Software Engineering Institute's maturity model.

The DSM managers were particularly interested in developing and implementing planning and measurement techniques. There was a sense of optimism that the software development process could be manageable and predictable, if only the work could be performed in a consistent manner and deliverables could be analyzed and measured. The DSM methodology provided a well-defined framework of the development process. Each task had pre-conditions (i.e., completed deliverables from previous tasks) and a clear objective (i.e., the creation of a deliverable used in the next task).

As DSM gained more experience with measurement activities, the managers allocated more of their resources to the quality assurance process. Several of the analysts were given the job of developing techniques for monitoring the development process (see Exhibit 5) and providing measurements of quality and productivity.

The developers were asked to provide more detailed information about the tasks they were performing. Programmers were asked to provide reports on the complexity of their programs. Designers had to include in their status reports, counts of interface complexity and evaluations of the degree of class reuse. Deliverables from database design tasks had to include measures of table normalization and transaction optimization.

While the information gathered from the measurement program was useful, the additional requirement of reports and summaries was adding to the workload of the developers. The extra work was minimal when the development task was small. For larger efforts, or for projects that had fallen behind

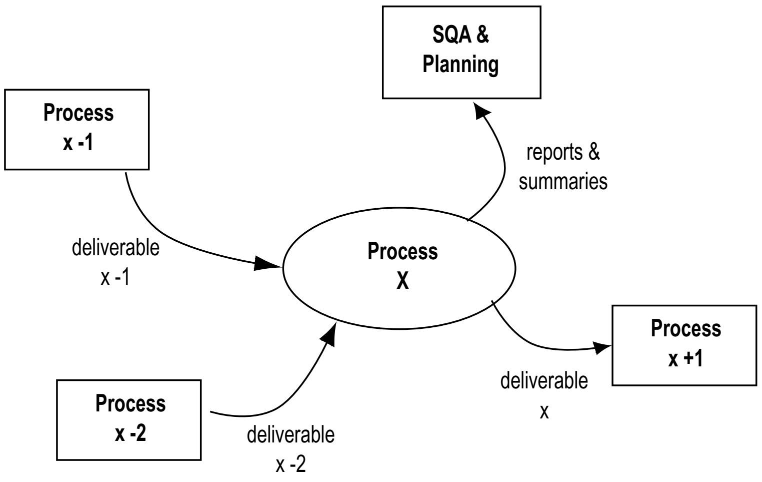


Exhibit 5 Generic Development Process

Exhibit 6 Sample Time Report

Timesheet week of: 5/14

Date	*Hours*	*Task Code*	*Description*
5/14	4	C12-4	Writing code for task 12-4
5/14	2	C12-9	Updating data dictionary
5/14	2	Misc.	Correcting requirements errors
5/15	8	C12-4	Same

Date: ____________________ Signature: ____________________
Date: ____________________ Approved: ____________________

schedule, the developers complained that the imposition was hindering their progress. It was not surprising that the developers took little care in the accuracy of the reports and summaries the developers prepared for the SQA and Planning analysts. After all, the real work had to get done and these reports were not helping them get the product out the door.

The DSM-DM worked well for the first year. There were no big projects, but lots of small ones. The Director of Applications Development allowed the developers to adapt and customize the methodology for each project so there were few complaints about redundant or unnecessary steps.

During that year, the development group adopted a new timesheet (see Exhibit 6) for reporting work hours spent on the various tasks defined in the DSM-DM. The report was easy enough. The Director and project managers would lay out a project plan. Each phase was subdivided into tasks with each task being identified by a code number. The methodology defined productivity as rate at which tasks were completed. The time required for each task was

Exhibit 7 Memo on Time Reports

To:	Development Team
From:	Director of Applications Development
Subject:	Time Reports

Just a reminder, do not charge hours to tasks after the phase review has been completed. Once a task is done and the phase review has been approved, hours need to be charged to the current task codes.

estimated and an expected resource requirement was recorded. Time was charged to tasks so that the difference between the expected resource requirement and what was actually charged became a rough measure of the task's status.

The project schedule was then computed by summing up all the task estimates and dividing that sum by the amount of time the developers were expected to spend working on the tasks. So a 100-hour task could be assigned to an individual dedicated to the task who would require approximately 2.5 weeks to complete it (assuming a 40-hour workweek). The same task, assigned to a three-person team, each person working half-time on the task, would charge 100 hours to the task after a week and three or four days (100/20 hr × 3 people).

During the first year of use, the Director noted a significant number of hours charged to tasks that were already recorded as complete. When he asked about it, the developers told him that they were just trying to keep the completed work up-to-date. For example, during a coding task, a developer might discover some ambiguity in the requirements statement. The time required to talk with the original analyst or to call the user had to be charged somewhere. The director thought it was strange to be working on a task that the development plan listed as complete. In addition, in most cases, the estimated resource requirement had already been exhausted, so charging more time to the task would make it look like the task took longer than it did. So the director set a policy of not charging time to closed tasks and notified the development staff by e-mail, as shown in Exhibit 7.

Time: Eleven Years Ago

Communication between the development group and the rest of DSM has always been difficult (see Exhibit 8). It seemed that the Director of Applications Development and the users were in a vicious cycle. The developers and the DSM managers had the most closely aligned views of a proposed software application at the beginning of a project. Of course, the application was not defined, so everyone could imagine anything they wanted. The development group would work through an initial phase of the DSM-DM. During this phase, the user's imagination and the vision in the developer's eye would evolve independently. At the phase review, it became clear how expectations had

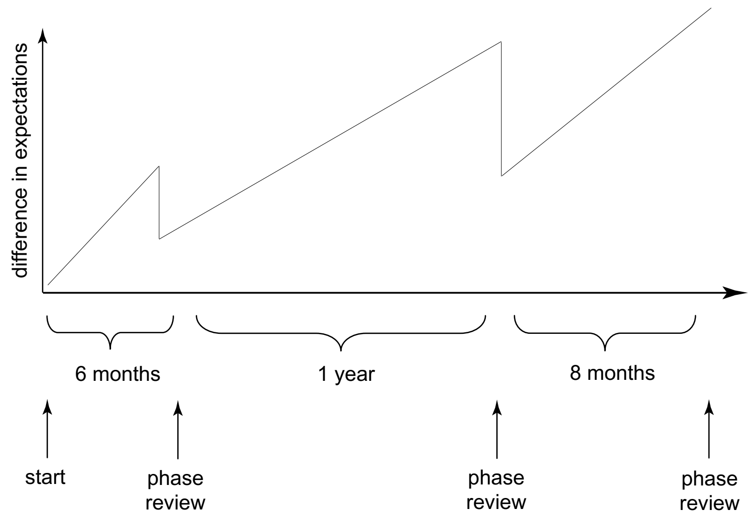


Exhibit 8 Growing Difference in Expectations

diverged. In conversation, everyone tried to explain his or her current understanding of the product. At the end of the phase review, expectations would be closer, but never close to perfect consensus.

The process was frustrating and many coped by practicing avoidance. They knew the communication process was imperfect and wearisome, so a manager would often delegate the responsibility of participating in the phase review, often with good reason; the person chosen to participate was usually closer to the operation than the manager, perhaps with a better understanding of the technology. But always, the delegate had less authority to make decisions. Phase reviews later in a project often required several meetings as the delegate needed time to return to her or his manager to discuss options and then come back some days later.

Time: Ten Years Ago

The first major development effort after the creation of the DSM Development Method was a product to support DSM clients. The Marketing group was reporting that the advice and support from DSM was useful, but some form of computer-aided system would be useful. Some major clients reported that other firms were promising software support within the year and Marketing was concerned that these clients would opt for competing firms if DSM did not introduce computer-aided support.

The CEO met with the Director of Applications Development and they agreed to make the DSM Strategic Integration Support System (DSM-SISS) product a priority and that it was a great opportunity to "do the job right" by strictly following the new DSM Development Method.

All the developers had received training in DSM-DM. The methodology had been used, in part, on several small efforts although the development

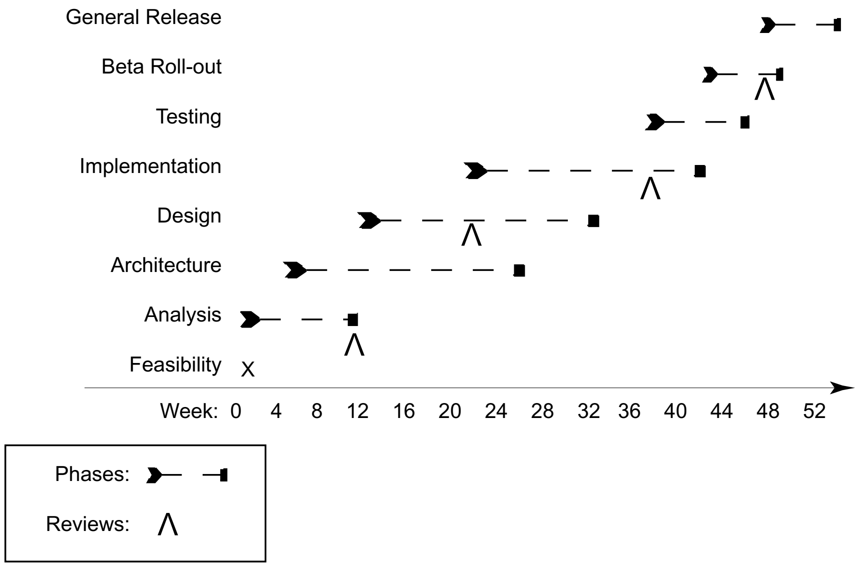


Exhibit 9 Overview: DSM-SISS Development Plan

staff had never been obliged to use the methodology strictly as written. The Director, confident that the development staff could deliver the DSM-SISS within a year's time, quickly initiated a project plan, laying out the phases and reviews necessary to bring the project in on time.

The DSM-SISS project used the DSM-DM to generate a project plan and lay out a schedule (see Exhibit 9). The analysts working on the measurements were able to produce impressive projections. When the DSM-SISS project was being planned, the measurements from past development efforts formed the basis for estimates of time and resource requirements.

The plan did allow for overlap of the phases. There was no need for a formal feasibility study because everyone agreed it was the best (and necessary) business decision and none of the managers in Applications Development felt there was any real technical risk.

Along with the schedule, the development plan outlined the staffing requirements for the project (see Exhibit 10). The first weeks of the project would require the services of three senior analysts. By the twelfth week, three designers would be added. The project would be fully staffed by week 28, with additional testers expected around week 45. The majority of this complement could be reassigned to other projects by week 52.

While the plan seemed consistent with the DSM-DM, one of the senior analysts sent an e-mail to the Director expressing some concern, who responded, as shown in Exhibit 11.

The analysts started working the requirements documents. They got good support from the regional managers who were very excited about the prospect of adding the software support tool to their mix of products and services. They were already talking (informally) with clients about the new system.

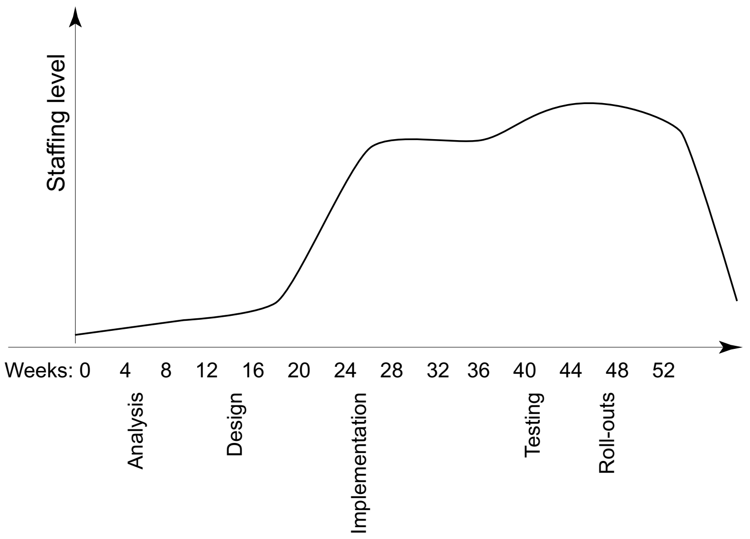


Exhibit 10 Staffing Requirements

Exhibit 11 E-Mail from Senior Analyst and Response

E-mail:

To: Director of Applications Development

From: Sr. Analyst

Subject: DSM-SISS Development Plan

I feel I must express some misgivings about the development plan we defined for the proposed SISS system. I can't help feeling we are creating our own reality. We think we know what the requirements of the system are, but we probably will not know for sure until week 12. If we discover by that point in time that the requirements for the system are much larger than we now expect, will we be obliged to work within the current time frame? It feels funny to be committing to a completion date when we are not sure of the amount of work we need to do.

The response:

To: Sr. Analyst

From: Director of Applications Development

Subject: DSM-SISS Development Plan

Thanks for your message. I know that planning and estimation are not exact sciences. If we discover the product is much larger than we anticipate, we will make necessary adjustments to the plan.

Exhibit 12 E-Mail after the Review Meeting

To: Development Team

From: Director of Applications Development

Subject: DSM-SISS Project

The project review of week 12 was not a great success. It is clear that we have a lot of work to do. Talking with many of the developers, I am confident that we can deliver a quality product in the time remaining. The analysts on the project have been directed to freeze the specification as it stands and to proceed with the Architecture and Design phase as defined in the project plan. Given our time constraint, we will look to streamline the tasks defined in the DSM-DM. Our next management review is schedule for week 23. Let's pull together and deliver a product we and our clients can be proud of.

However, every interview with a regional manager or client identified additional features and conflicting priorities.

Shortly before the first formal review, the analysts were going crazy trying pull together some coherent, consistent statements of system requirements, but at the review meeting it was clear that the requirements were not close to being complete, the scope of the project was far greater than expected, and the frustration among the analysts was unmistakable. The executives at the meeting refused to consider extending the target date for the product. The VP of Marketing insisted that the product must be available on schedule or they would begin losing important clients to the competition.

After the review meeting, the Director of Applications Development sent out the e-mail shown in Exhibit 12.

At this point, the number of developers working on DSM-SISS increased as designers and programmers were assigned to the project. The development team took the memo from the Director as a license to ignore aspects of the methodology they found inconvenient. After all, the goals had been clearly communicated: get something out the door by week 52 — although the product is much larger than originally envisioned and the project plan remained unadjusted.

Managers were a little concerned as they heard the company's development methodology, DSM-DM, being referred to as "diz-dumb." But none of the managers publicly objected to the sarcasm because they knew they were working long hours and were pulling together as best they could to accomplish a difficult objective.

A few days before the scheduled week 23 review, the Director of Applications Development met with one of the regional managers. The Director pulled up the Event Definitions, Class Models, and Database Schemas for the system and was prepared to ask questions he had received from the development team. But before the Director could begin describing the various development models, the regional manager told the Director that several of DMS' clients had already received beta versions of similar products and that the features that seemed to be the most valuable were not even included in the original requirements statements. The regional manager said he wanted to discuss the new features at the review meeting and was wondering if the Director could prepare some estimates for what it would take to include them in DMS' product offering.

The Director, trying to do the best job he could, arranged to postpone the review meeting for one week and met with his managers and several of the senior analysts to prepare some response. They projected the new features would increase the size of the system by about 30 percent, revising the schedule to show the product rolling out about week 65.

When this revised plan was presented at the review meeting, the executives were not pleased. The CFO said she was uneasy about continuing the project. She quoted the total in salaries and overhead consumed by the project to-date and wondered aloud what this money was buying. The VP of Marketing reasserted that the product was critical to the success of the company and that if more people were needed to get it done, then more resources should be allocated. The Director of Applications Development pulled out the Event Definitions, Class Models, and Database Schemas for the product to try to explain the status of the project, but most of the people in the room were unimpressed. The CFO's comment was, "These don't mean anything to me." The Director responded by pointing to the DSM-DM to support his position, but this did not carry much weight. The CEO saw that the meeting was going nowhere and ended it by saying he would be meeting with people individually to work out the next step.

The next week was a long series of meetings and hallway discussions. The next Friday, the CEO called the Director into his office. When the Director arrived, he saw that the VP of Marketing was already there. The Director was told that for some time now, the company had been looking into the possibility of buying out a small private company in the Midwest. The owner was retiring and was interested in selling his business. The Director recognized the company as one that had a nice local niche, but never competed with DSM outside the northern Midwest region.

The VP of Marketing reported that Midwest had a product that was exactly like the one DSM was building, and it was already in use by clients in that region. He was recommending that DSM buy out the Midwest company and market the product under DSM's name. The CFO had already reviewed the finances and her opinion was that the purchase would be good for DSM. The CEO turned to the Director and said he wanted him and one of his analysts to fly out and review the Midwest product and come back with a recommendation.

Before leaving, the Director told his development team about the move to buy a competitor's product and instructed them to stop pulling overtime for a few days and to turn their attention toward completing the documentation and cleaning up the work they had completed so far in preparation for whatever comes next.

The Director and the analyst did not want to find a workable product at Midwest, but they did. They found a very different development environment

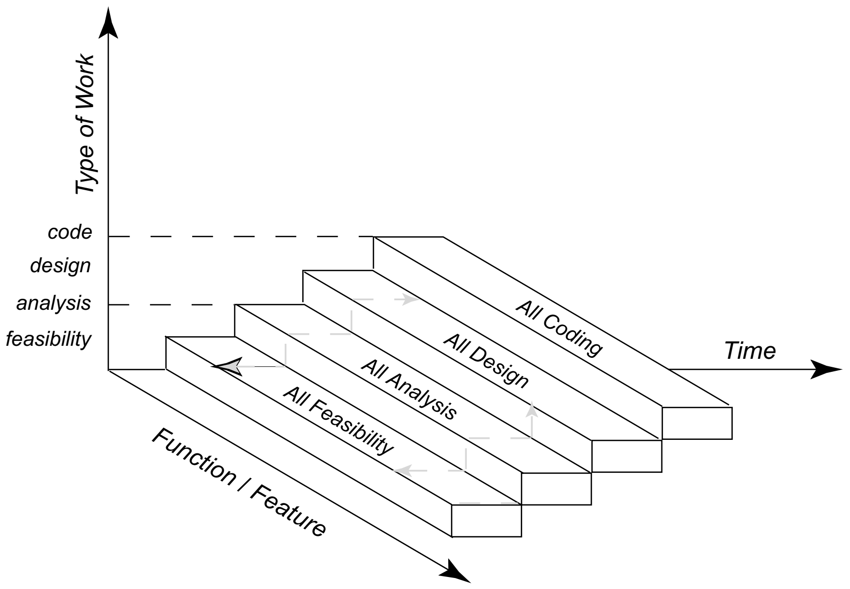


Exhibit 13 Methodology as Steps

and a software system that performed well with most of the needed features. Better yet (or worse, depending on your point of view), the Director and the analyst came to believe that the Midwest product was functional now and that missing features could be added in subsequent releases. The Director reluctantly called the CEO and recommended that DSM proceed with the purchase and that Marketing should go ahead and start planning for the wider release of the Midwest product as DSM-SISS.

While the Director was at Midwest, he and his counterpart had several interesting (if not philosophical) discussions. Despite the circumstances, the two had a lot in common. One day, they were having lunch in the conference room and began sharing the impressions of the state of software development. The Director drew a graph on the whiteboard with one axis labeled "type of work" and one labeled "time." He commented how bizarre it seemed to break up the *y*-axis by phase — feasibility, analysis, design, etc. He commented: "When I work on software, I am thinking about all these things at once … but one feature at a time."

As they talked, the Midwest manager drew a third axis to represent the features or functions of a proposed system (see Exhibit 13). This made the Director even more irritated. "This drawing suggests we are trying to gather all information about feasibility in phase one, all information about requirements in phase two, etc. This is crazy! We can't be that sure about this information. By the time we get to design and coding, the lower steps have crumbled."

As the two finished lunch, the director erased most of the drawing and began doodling. "If we could manage in a manner compatible with the way I work, we'd recognize that software is built in clumps. We build a cluster of functions — performing analysis, design, coding, and testing on them. At a

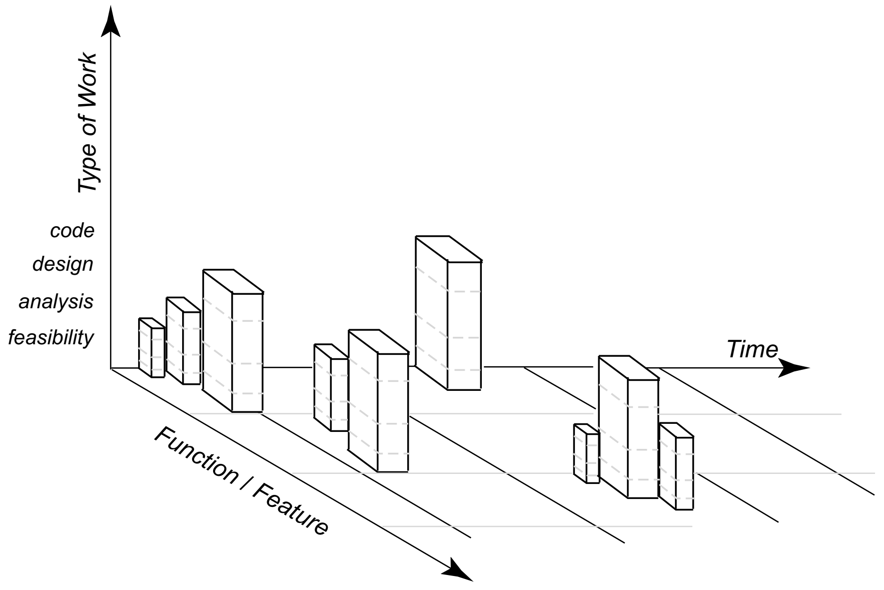


Exhibit 14 Methodology as Blocks

later point in time, we build more software. At some point we may find a need to enhance the original work. At the same time someone else is adding another feature — doing analysis, design, coding, and testing (see Exhibit 14).

The Director's counterpart commented that that was pretty much how his group worked. "It has the advantage of always having a demonstrable system." The Director flashed back to the review meeting. He stood there, feeling rather stupid, with his database schema and class diagrams, unable to show that any real work had been done. The work was real but not in a form that could be appreciated by the decision makers.

They agreed to continue the discussion. The Director left Midwest with a sense that the two had a lot to learn from each other.

Time: Nine Years Ago

After the decision was made to use the Midwest software rather than continue with the in-house development, the managers in Applications Development conducted a post-mortem on the project. They knew that the developers were upset and viewed the whole effort as unfair.

During one of the review meetings with the development staff, one programmer said: "We all feel let down. I went back over my timesheets and the project status reports. We were doing all the right things. Our tasks were coming in almost exactly on budget. We were completing tasks almost exactly on time. So why did we do all this work only to have the rug pulled out from under us?"

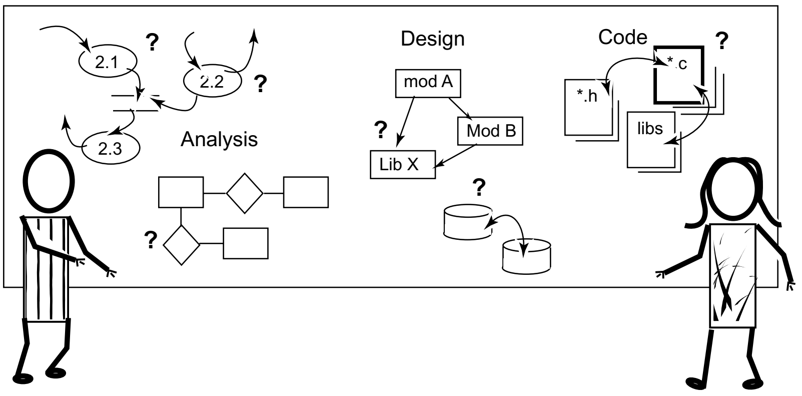


Exhibit 15 Methodology Representation

Later, the Director was talking with one of his senior managers: “The programmer is right. How is it that we could look so good on paper and not know that things were so out of control?” “Well,” replied the manager, “I think we all know in our heart of hearts that the information on the time sheets doesn’t reflect reality. It is as if we ask the developers to tell us what we want to hear. The fact that 100 hours was charged to a 100-hour task does not mean the work is done. Having a review where everyone signs the requirements approval form does not mean that we know what the user really wants nor what they should have.”

The Director and the manager went into the conference room. The Director drew some symbols representing the DSM development methodology on the whiteboard (see Exhibit 15). The manager pointed to the “code” phase and said: “I think a big problem is that, if we are hit with a change to the system (code or design or requirements) at this phase, we don’t have a plan for rippling back through the phases to bring everything up-to-date. In fact, we don’t even know what might be affected by a given change. So the developers either go hunting through all the past work looking for what needs to be updated, or (more likely) they just change the code and forget about the other stuff. In fact, that policy about time reports tends to encourage just working on the current task because they can’t charge time to work we thought was done.”

The Director remembered some of the conversations he had had at Midwest. One of the Midwest developers told him that they did not fill out time reports. He thought that was strange but did not think much of it at the time. He also remembered sitting in on a group session the Midwest developers had in their war room (see Exhibit 16). The Director erased the whiteboard and drew a mock-up of the war room. They had diagrams and printouts hanging all over the walls. Each section of the wall had material from different documents.

The Director continued, “They were talking about a change request. I don’t remember the specifics but I remember thinking they were talking about a significant change. One would point to a database schema and list the changes

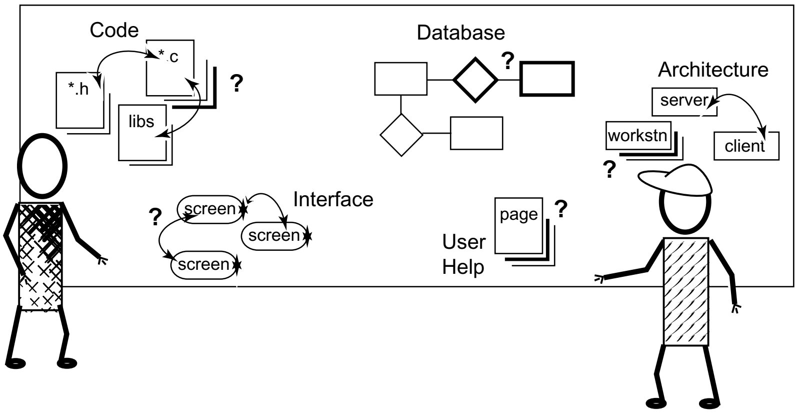


Exhibit 16 Midwest War Room

that might be necessary. Another would interrupt, pointing to a piece of code and marking it as "needing review." A third would draw some changes on an interface diagram, saying that if the table had to change, this interface should be altered. After a short time they had outlined changes to every document affected by the change. They agreed to get back together the next morning. I assume they left to make the changes. I thought it was strange."

The Director and manager agreed that some changes were necessary before the next big project was started. "They also said they had no projects," the Director said to the manager as they left the conference room. "I'm not sure what they meant by that."

After the DSM-SISS project's failure, managers and executives of DSM wanted to know how the estimates could be so far off. The analysts got information about the Midwest product and created a set of numbers based on its actual features and code. They discovered that the Midwest product was twice the size of the product described by the initial requirements statement of the DSM-SISS (two and a half times the number of function points). They reported that the features the marketing group were trying to add to the project during the first six months would have doubled the estimate for the project.

The SQA (software quality assurance) analysts concluded their report by saying that the original estimates were correct, given the projected size of the product, but that the requirements statements were not accurate. The Director of Applications Development asserted, with a great deal of frustration, that the development team could not have known the full extent of the system's requirements. That knowledge was not available until they and the regional managers had a chance to refine the customer's view of the product.

One of the post-mortem meetings concluded with the CEO saying: "So, it appears we can accurately measure our software products after they are finished. We can accurately say what a project should have cost us if we had known everything at the beginning of the project that we know at the end. But we can't know at the beginning of a project what a project will require until the end." He shook his head and said, "Work on it."

Time: Eight Years Ago

After the disaster of the DSM-SISS project, the development department entered a period of adjustment. Most of the DSM-DM fell into disuse. Project managers were allowed to pick and choose those tasks that seemed most appropriate. As time went on, fewer and fewer of the tasks were considered "appropriate."

The support and enhancement of the DSM-SISS remained with the Midwest group. The development on the West Coast continued with other independent products but more and more time was spent on maintaining and upgrading existing systems.

Over the next year or so, the CEO saw that various projects were initiated. He reviewed the accounting of the money being spent on these projects and convinced himself that things were moving in the right direction. But direction was not always without its problems.

The CFO was very uncomfortable managing a division that was increasingly answering to other managers. She recognized that the Director of Applications Development was in a difficult position as he tried to coordinate development in support of many parts of DSM. Increasingly, conflicts developed, such as the one between Finance and Marketing. A project to develop customer relations information for the regional managers had been approved. It was expected to take six months to complete. The CFO accepted a delay in a proposed upgrade to their customer credit system, expecting the work to begin after the marketing system was completed. After about five months, it was clear that the marketing system was not going to be completed on time and the capital management system would be delayed even further. The Director of Applications Development suggested that some of the developers working on the marketing system be reassigned to start the customer credit project. This meant that the marketing system, already behind schedule, would be even later.

The CFO resented having to spend time on this conflict. From her perspective, she was being generous to support the marketing systems at all, and now to have her generosity repaid with delays in critical systems was hard to take.

The Director went to his developers and asked for an extra effort to get the marketing system "out the door" as soon as possible. The developers responded by putting in a significant number of overtime hours. They also were liberal with the interpretation of requirements; after all, it was clear that the goal was to get "the system" out the door on time. The exact definition of "the system" had always been nebulous. So the fact that extensive error handling was not implemented was technically not a violation of requirements because it was never explicitly stated. Cutting corners on the rigor of data interfaces seemed justified because it allowed them to deliver "the system" on time.

They succeeded by the measurements that were most prominent. The system was put into production only two weeks late. The marketing people were a little disappointed that some of the features they thought would be there were not, and that the training and conversion support they thought

Exhibit 17 Creation of Steering Committee

To: CEO

From: Director of Applications Development

Subject: Project prioritization

The applications development department needs more information about the strategic contribution computer applications are making to DSM. We have a long backlog of system development requests. Our effort to prioritize the requests requires more information than we have. The decisions we make about which projects take precedence will never be optimal without input from all affected DSM divisions.

I request that a steering committee be created with representatives from Finance, Marketing, Operations, and International Divisions. Their charge would be to approve and prioritize development requests and act as overseer and facilitator of relationships between the development group and its users.

cc:

CFO

they would have was rather sketchy. The developers were happy to have their weekends back and they began working on the customer credit system.

As time went on, Marketing began sending more and more maintenance requests. They had discovered that it was easier to get maintenance requests in the queue than "new development" requests. But the Director noted that some of the maintenance requests were larger than the original new development proposals. As he assigned people to deal with the maintenance jobs, there were fewer developers working on the new systems.

The Director of Applications Development knew that the situation was not going to get better. He reasoned that as long as the managers requesting services were outside the decision-making process, there was going to be dissatisfaction with any decisions made. He talked to his boss (the CFO) and to the CEO, suggesting the creation of a steering committee (see Exhibit 17).

The Director had a conversation with the CEO in which the Director expressed his hope that the steering committee could exercise a degree of authority in mediating the decision-making process. The CEO agreed and called the first meeting of the committee where he established their mission (see Exhibit 18).

The first few meetings of the steering committee were attended by the executives themselves. Their initial work seemed promising; they made decisions about which of the development requests should have priority. They were even successful at merging several related requests by identifying overlaps in the subject matter and assigning subject matter experts from different divisions to work jointly on an integrated system.

The communication process seemed to improve over the first years of the committee's existence. With representatives from all major domains of the company, decisions were more open and people were aware of any conflicting issues. The steering committee developed a good sense of the projects being

Exhibit 18 Calling First Meeting

To: DSM Officers

From: CEO

Subject: Applications Development Steering Committee

I am forming a new steering committee to address the growing demand for a computer application system. All divisions of the company have identified the need for new applications supporting both internal functions and our clients. It is clear we cannot accommodate all the requests at once, so a better method of prioritizing application requests is in order.
The committee will be made up of:

- CFO — (who will chair the meetings)
- VP Marketing
- COO
- Director of Applications Development
- A representative of the international divisions

I will be a non-voting member of the steering committee.

The charge to the committee is to review project proposals, review information from the requesting organization to assess potential benefits, review information from the applications development department to assess resource requirements, approve and prioritize the project requests, and monitor the project's status.

The first meeting of the steering committee will be next Thursday, 10 a.m.

proposed and on many occasions found ways of merging proposals from different user areas into a single project serving shared interests.

A major problem developed, almost unnoticed. The size of the approved projects became larger and larger and the resource estimates became increasingly unrealistic. The steering committee began to view small proposals as less important than "significant requests." The small jobs only delayed and complicated the "real" work of the development group. This trend was exacerbated by the long lead-times between a project's request, a project's approval, and the completion of the work. The steering committee met only once each month. There was always a backlog of work and decisions were often delayed over several meetings. Most of the committee's time was spent on the larger proposals. People submitting project requests quickly learned that bigger proposals got preferential treatment. Smaller proposals took too long to be considered worthwhile.

After the first year, the executives began sending their subordinates in their place. This added to the lead-times on proposal considerations. Inevitably, their delegates would be faced with issues they felt they could not answer. They would return to their divisions to discuss the matter with their superiors and report back at the next meeting. This doubled the time it took to consider and act upon a development request.

By the time DSM was considering the DSM-SISS, the steering committee had become slow and unresponsive, leaving the Director of Applications

Exhibit 19 Changing the Reporting Relationship

To:	All DSM Employees
From:	CEO
Subject:	Chief Executive Officer

It is my pleasure to announce the creation of a new executive position of Chief Information Officer. I am naming the current Director of Applications Development to fill this position. All functions under the current Director will move out of the Finance department into the newly created division. The office of the CIO will report directly to me.

I, and the other officers, feel that the new position will strengthen the role applications development has played in the growth and success of this company. The Director has proven to be a valuable asset to the company's strategic planning and we are confident that he will continue to serve us well in his new position.

Please take the time to congratulate the new CIO on his promotion.

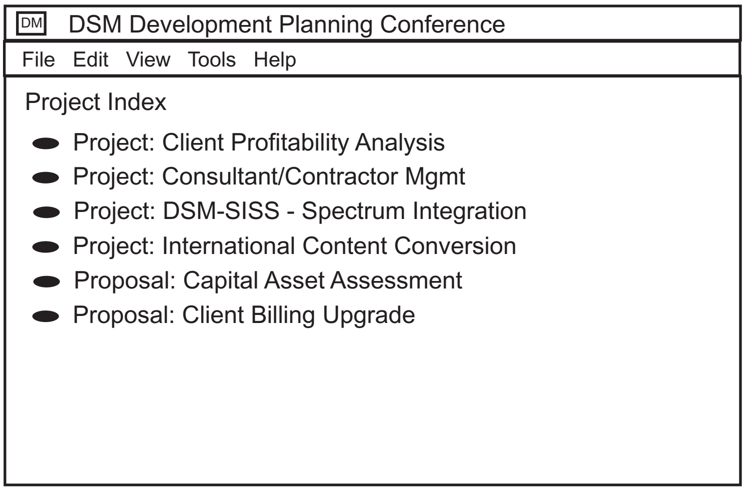
Development alone to coordinate the many considerations and demands of the proposed system. The CEO stepped in to support the director. The CEO even began making unilateral decisions when the decision was on the critical path.

The close working relationship between the Director and the CEO, and the company's experiences during the DSM-SISS project, convinced the CEO to name the director to the position of Chief Information Officer, and to change the reporting relationship (see Exhibit 19). The Director achieved the formal status he had earned over the years. The development organization now reported directly to an executive-level officer.

The new CIO began laying the groundwork for ending the steering committee. He worked with his developers to define a browser-based conference to access the development plan databases (see Exhibit 20). The CIO began by announcing that the current project status reports were available on the intranet and that he would not be reviewing them separately during the steering committee meetings.

Once members were accustomed to the use of the conference to reference project status, the CIO introduced a discussion forum where project proposals were posted (see Exhibit 21). Members were encouraged to review the proposals online and post questions and comments (see Exhibit 22). When the CIO determined that a consensus had been reached, he posted the summary in the discussion and asked if everyone concurred. He would then review the discussion and the result at the next steering committee meeting. Quickly, the members began to rely on this discussion and decision-making forum.

When the CIO was convinced the group was functioning well using the conference, he suggested to the CEO that the steering committee meetings be canceled and that the group do all future business through the online conference (see Exhibit 23).



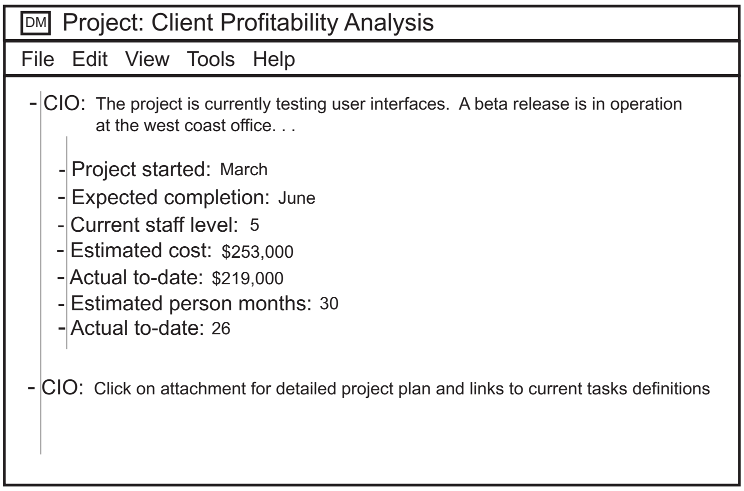
DSM Development Planning Conference

File Edit View Tools Help

Project Index

- Project: Client Profitability Analysis
- Project: Consultant/Contractor Mgmt
- Project: DSM-SISS - Spectrum Integration
- Project: International Content Conversion
- Proposal: Capital Asset Assessment
- Proposal: Client Billing Upgrade

Exhibit 20 Shared Project Repository



Project: Client Profitability Analysis

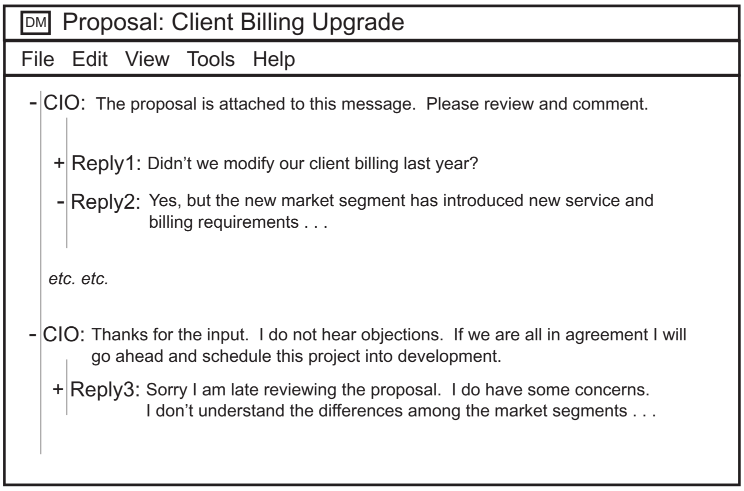
File Edit View Tools Help

- CIO: The project is currently testing user interfaces. A beta release is in operation at the west coast office. . .
  - Project started: March
  - Expected completion: June
  - Current staff level: 5
  - Estimated cost: $253,000
  - Actual to-date: $219,000
  - Estimated person months: 30
  - Actual to-date: 26
- CIO: Click on attachment for detailed project plan and links to current tasks definitions

Exhibit 21 Project Status Page

The CIO continued to meet regularly and informally with all the decision makers. However, the flexibility of the online conference allowed him to conveniently invite other experts into the discussion. One proposal involved a complex interface. The CIO invited an outside expert to post information to help the steering committee assess different options.

When the Director of Applications Development was named the CIO, he took some time to think about his role within DSM. His personal goal was to identify the type of information each of his constituents needed. The CIO reasoned that if he could provide the information that each colleague expected, they had a good chance of maintaining good communication, effective decision



Proposal: Client Billing Upgrade

- CIO: The proposal is attached to this message. Please review and comment.
  + Reply1: Didn't we modify our client billing last year?
  - Reply2: Yes, but the new market segment has introduced new service and billing requirements . . .

etc. etc.

- CIO: Thanks for the input. I do not hear objections. If we are all in agreement I will go ahead and schedule this project into development.
  + Reply3: Sorry I am late reviewing the proposal. I do have some concerns. I don't understand the differences among the market segments . . .

Exhibit 22 Proposal Discussion Area

Exhibit 23 Cancellation of Steering Committee Meetings

To: DSM Officers

From: CEO

Subject: Applications Development Steering Committee

I am canceling all future software application steering committee meetings. It seems the new on-line conference system is working and there is no longer a need to coordinate our schedules and meet regularly for the purpose of reviewing project requests.

making, and more reasonable prioritization of goals (see Exhibit 24). If he provided the wrong information, he was sure his effectiveness as an officer of the company would be severely compromised.

The information relationships with other officers of the company would be one of "service." That is, the office of the CIO would be judged by how well it contributed to the success of the other divisions of DSM. So, the CIO was determined to keep the other divisions of DSM informed of the status of their particular requests. He also knew that he would always have more requests for service than he had resources to fulfill them. The information he shared with other executives would have to include criteria and strategic goals he used to prioritize development requests. It would not be enough to say to the VP of Marketing that the request for an update of field support applications would not be addressed for six months. The information would also have to include what applications were given precedence and why. The decision-making process would have to be coordinated and flexible, and would need the support of his boss, the CEO. The information shared with the CEO would have to be tailored to help the CEO direct the discussion of corporate priorities.

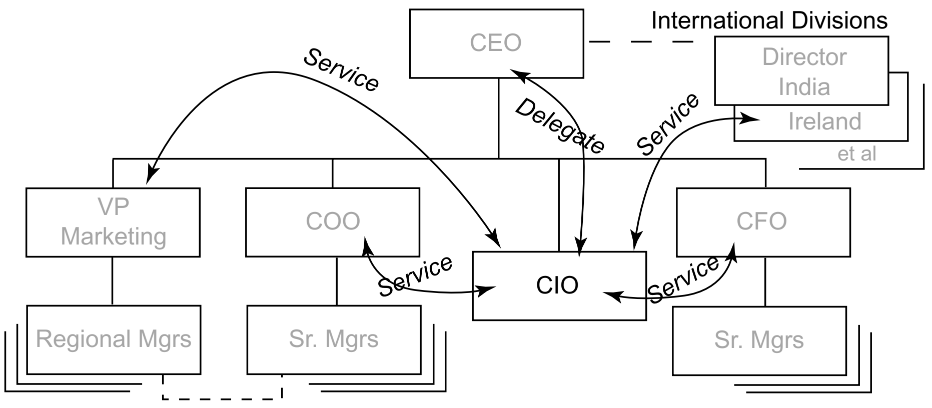


Exhibit 24 Communication across DSM

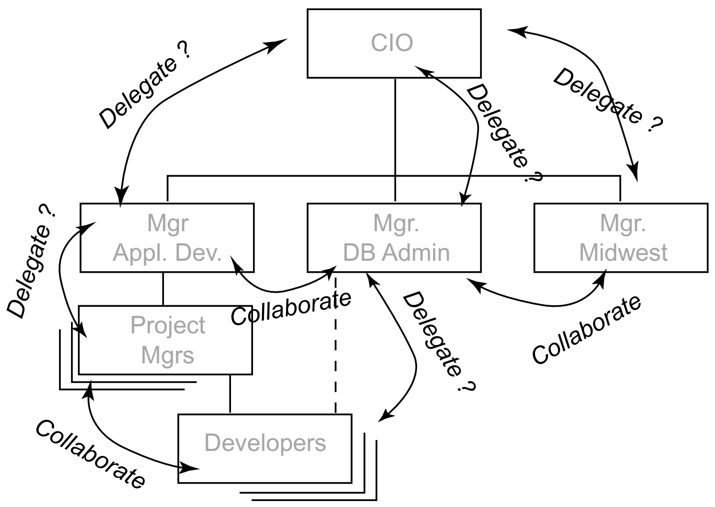


Exhibit 25 Communication within the Development Group

The CIO determined that communication within the development group would be mostly collaborative (see Exhibit 25). While there were formal administrative relationships, traditionally, managers have tried to function as a member of the development team. The CIO wondered if that was really the best arrangement, but every manager had come up through the ranks of the development staff and enjoyed staying involved with the technology and the development work. For these communication channels, the CIO determined to make the development plans and all elements of the environment open to debate and refinement. Decisions about standards, methods, practices, tools, and training needed the perspective of developers. Of course, decisions had to be made and it was clear that the managers would have that responsibility, but all decisions had to be informed by the lessons learned by the developers and project leaders.

Exhibit 26 E-Mail from Midwest Developer

To:	Midwest Development Manager
From:	Concerned Developer
Subject:	Upcoming DSM Development Conference

I am really concerned about the upcoming conference. I see no reason for our parent company to impose its formal and stifling methods on us. After all, it was our product that got them out of a bind several years ago. How should we respond to this?

Reply:

To:	Concerned Developer
From:	Midwest Development Manager
Subject:	DSM Development Conference

I think this is a good chance to build some bridges and expand our own influence in the larger organization. We should go with an open mind. We will take what works and change what doesn't. My advice to all of us is to take every opportunity to show the good work we are doing. Make as many contacts as you can and speak out in favor of the techniques we use in building successful products. I think the people at DSM are open to innovation.

By this time, it became increasingly clear that the products being developed by DSM and Midwest had to be updated and integrated. The Director, now the Chief Information Officer, had wanted to bring the best of both development groups together. He felt it was time, at long last, to do something significant and positive. He initiated his campaign by convening a development conference. A week was set aside for all developers to gather in San Francisco to come up with a plan for developing closer working relationships. Sessions were planned to provide the Midwest group with detailed orientation to the DSM-DM methodology, the current set of standards used on the West Coast, and the development environments currently in use.

When the developers at Midwest received the announcement of a "conference," they immediately concluded that DSM was finally going to indoctrinate them into its stiff and formal methodology.

One of the Midwest developers e-mailed her boss, and the manager of the Midwest development group replied, as shown in Exhibit 26.

So the conference took place. Much of it was boring. The two sets of developers began taking longer and longer lunch breaks to exchange stories about past projects. They even convened impromptu workshops to learn each other's development secrets. Both the DSM CIO and the Midwest manager agreed that the informal exchanges were probably more valuable than the planned sessions so they cut short the formal training in favor of the more spontaneous sessions.

During one of the breaks (i.e., spontaneous sessions), a large group was gathered in the conference room discussing where the application development industry had come from and where it was going. One of the DMS

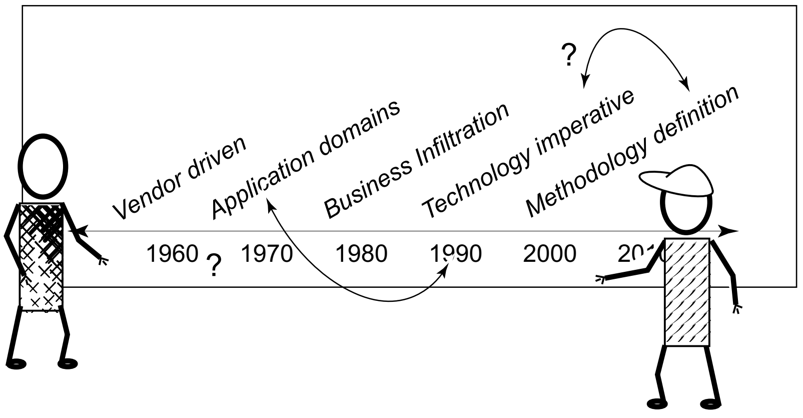


Exhibit 27 Development Eras

developers went to the whiteboard and drew a timeline, saying he was happy to see the passing of the era of vendor-supplied bundled applications. Someone pointed out that maybe things had not changed that much. The big ERP implementation projects are painfully reminiscent of those days — maybe the era never ended (see Exhibit 27). Someone else grabbed a marker and wrote "Application domains," saying she was glad we are developed beyond data silos. A voice from the back said, half jokingly, "Hey, those were some of my best work."

One of the project managers wrote "Business infiltration" to represent the shift from discrete systems to systems that better integrated into the daily business operations. That prompted one analyst to write "Technology imperative." "It seemed that along with the business integration came the sense that technology was the end-all solution to everyone's problems. Everyone has to have the latest version and the most advanced hardware. I don't think the latest version necessarily adds value." There was overlapping debate over which era came first and if any really ended. At one point, there were five people at the board, each armed with a marker.

"What about development methodologies?" came a voice from one of the junior developers. No one had wanted to touch the subject of methodology, but apparently the kid did not know any better. The CIO walked up to the board and wrote "Methodology definition." This legitimized the subject, so people began to talk.

The group began exchanging ideas about how development projects should be partitioned into manageable steps (see Exhibit 28). Some were dutifully defending the phased approach, while others began to assert there were major problems with the construction or the factory analogy. After a couple of minutes, the whiteboard had been erased and a new image was being drawn.

It was clear that the general consensus was that project methodologies seemed to impose a feeling of an assembly line and that image did not feel good to most in the room. The drawing on the board took on the mantle of satire. The conversation began to slow.

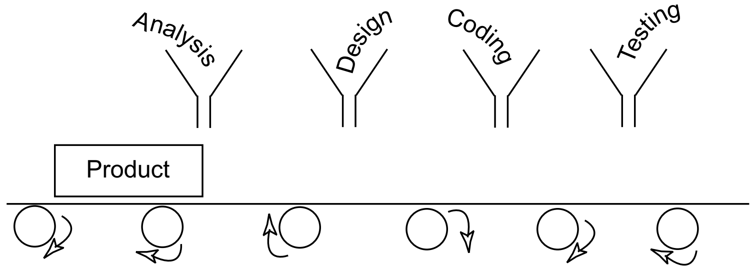


Exhibit 28 Methodology as an Assembly Line

The Midwest manager stepped forward. "What if we turn things around?" he said, erasing the board again. "The model we have been discussing seems to divide the work of a project by the type of work first, then the individual features, functions, or objects. Maybe we can divide up the development effort by the object, function, or feature first and then subdivide that by the type of work — requirements, design, coding, etc."

He began to draw a huge cube that filled the whiteboard. All the lines were dashed except the lower left-hand corner. "Let's say the box here is the product. It is mostly undefined. We can imagine the whole thing but only in our imagination."

"Down here," referring to the solid lines, "we have what we know for sure. We know enough to define a first version — say chunks 1 through 5. We think we can add chunks 6 to 8 in the second version. We might even be able to foresee improved versions of chunks 2, 4, and 7 — but we won't get to them until a third version. Of course, these future versions will actually be determined by the feedback we get from the users when they start using the first version (see Exhibit 29).

The CIO recognized the drawing from the Midwest conference room conversation several years ago. Obviously, the manager had spent a lot of time thinking about this. "So, what are these blocks?" asked the CIO. "They are features or functions or scenarios or requests," replied the Midwest manager. "They are cohesive chunks of the product that the user has requested."

"But what work products do each of the blocks represent?" insisted the CIO. The Midwest manager replied, "They represent our understanding of the user's requirements (written in the form of use cases and object definitions), our declaration of how the requirements will be implemented (in the form of database tables and code/class structure), the code satisfying the requirements, the test scenarios used to exercise the product, the user training updates, and help systems reflecting the new enhancements. Each version adds value/functionality and keeps all the information current. This way, the objective is not to move a complete system along the assembly line as fast as possible. The objective is to keep adding value — piece by piece — forever."

There was silence in the room. The CIO was recalling the conversation he had had a few years earlier with the Midwest development manager. After a moment, the CIO said, "Well, OK then. Let's work on it."

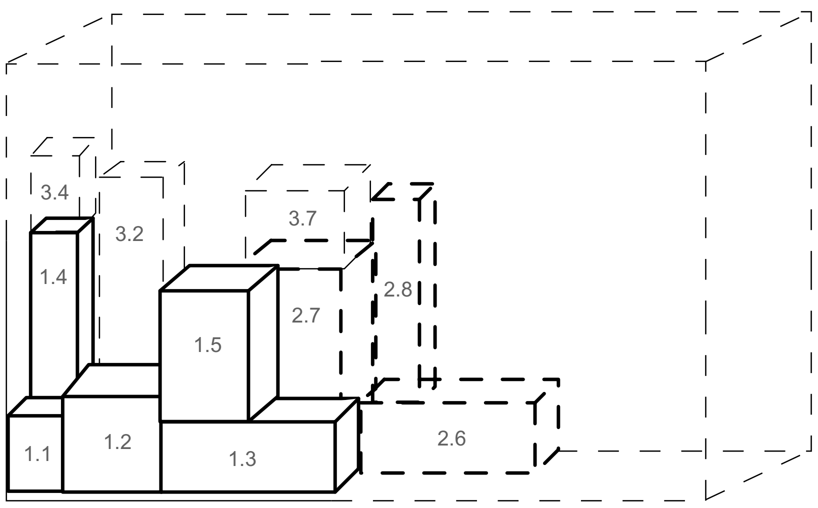


Exhibit 29 Methodology as Continuous Build

The developer's conference was notable for two reasons: (1) the developers from headquarters and the Midwest left with a sense that they were all in the same boat; and (2) the developers knew that their managers were interested in improving the way software was built at DSM and were willing to listen to their ideas. After the now-famous conference room discussion, many hallway discussions occurred. By Thursday, there was a consensus that however they redefined "process" and "phase," they needed a common repository to store their "stuff." The exact definition of "stuff" was elusive, but a shared memory was going to be critical to their success. In the minds of the West Coast developers, this was visualized as a great knowledge base where all the project deliverables was stored. In the minds of the Midwest developers, this was visualized as a database storing their collective memory.

As they talked, a simple model emerged (see Exhibit 30). They all agreed that the products they built consisted of many different kinds of elements (e.g., pages of requirements, pieces of designs, code, libraries, user manuals, database tables, interface designs, test cases, etc.). They all agreed that they worked by identifying some objective (i.e., a task) and building the elements as they work toward that objective. But, interesting insights began to emerge as they discussed the elements themselves. Elements were connected — not to other elements of the same type, but to elements of different types that described the same feature or subsystem. The "cohesive chunks of the product" that the Midwest manager drew in the conference room were strands of pieces of knowledge about a function.

Before they left the conference on Friday, the developers had added a notation to their informal model to represent the associations among elements. They also decided there had to be some agreement as to what those elements are and what associates were typical, so an entity called "element definition" was added to the model.

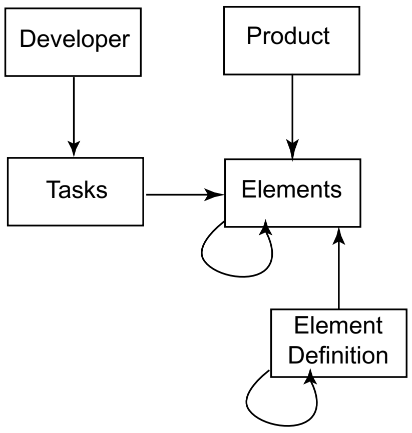


Exhibit 30 Initial Information Model

The action item accepted by every developer was to begin defining each model in greater detail and to start populating the models with actual instances of elements and element definitions. No one was sure where this would lead, but all agreed it would be an interesting journey.

Time: Seven Years Ago

Over the course of the next year, the CIO deliberately defined development efforts involving both West Coast and Midwest developers. Many of developers at DSM headquarters spent time at the Midwest facility (see Exhibit 31). Part of every project plan was the enhancement of the repository model and improvement of the development practices using it.

The developers visiting from the West Coast got a real appreciation for the connection between physical surroundings and philosophy. Each member of the development staff at Midwest had his own office. There were a couple of workstations set up in the common area where users and developers had frequent informal meetings to discuss the latest features and discuss plans for the next round of improvements. But what intrigued the West Coast developers most was the time spent in the “war room.” The war room was a conference room dedicated to the development group (i.e., they had exclusive use of the space). It was used as a command center. The walls and whiteboards were thick with diagrams and notes and code printouts. Several times a day, groups of developers would confer about the current project. One of the Midwest developers commented that it was great to be able to walk into the war room and immediately have a sense of the group’s status.

By contrast, the Midwest developers who spent time at the West Coast facility noted that the partitioned layout seemed to dictate a partitioned approach to work. With no space to exchange information informally, the developers had to pass information from person to person in a more structured way.

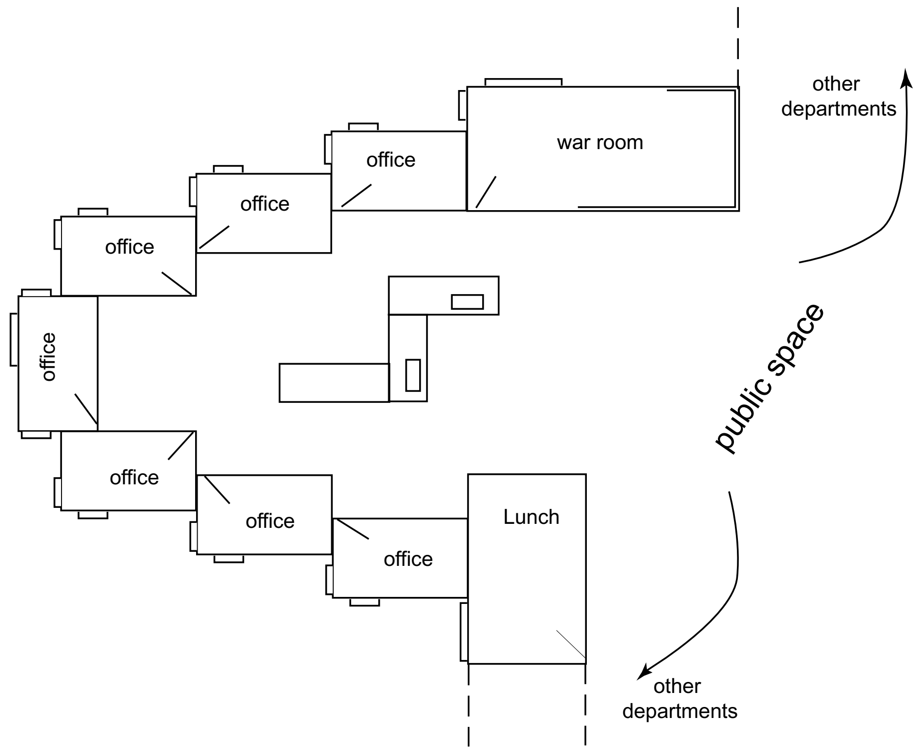


Exhibit 31 Midwest Development Work Area

The developers set up an online conference with a discussion section to share their ideas on the repository and how it should be used. Within six months, they had put together some prototypes of a database system for the repository and were updating it with their element definitions. As they refined or added definitions to the repository model, the developers noted many common element definitions. Each shop included elements such as:

- Test cases associated with units of code
- Requirements associated with object/entities
- Object/entities associated with database tables

There was disagreement as to what constituted a unit of code. The West Coast used mostly C. Midwest used C++ and some Java. There were differences in the format and size of the "requirements statements." The West Coast used a variation on the formal DSM-DM with defined event and data definitions, and the Midwest requirements read more like narratives. The West Coast documented business rules associated with data object definitions, while Midwest recorded its policy rules with the requirement narratives.

The greatest disagreement involved the definition of "design." Both groups used different methods and models to represent design decisions. The West Coast made use of state models for interface designs while Midwest used less-formal navigation diagrams. Both used class diagrams but they used different graphic notations.

Vocabulary was also a difficult issue. Each group had its own definitions for common words. "Object" and "Entity" were synonyms at Midwest, but they had distinct definitions at DSM headquarters. Midwest used "use cases," which was very similar to what the West Coast called "event definitions." The manager at Midwest and the CIO often got together and declared company definitions when consensus was not reached quickly.

The results of the effort were rather impressive. While the development efforts initiated since the conference were small (i.e., none longer than four months in duration with four to eight developers), the work being completed was good. The users liked the fact that real solutions were occurring, often with an apparent improvement in overall quality. The developers loved building good applications as well as building effective development techniques. They also liked the idea that they were significant players in defining the methodology although no one used the word "methodology." One developer said it felt like they were creating a culture — not a method.

There were problems. The biggest was with West Coast managers. They were feeling uneasy about the lack of control. They complained to the CIO that there was no good way of measuring productivity and monitoring progress. "Users ask for something and the developers give it to them," complained one manager. "We need a way to bring back some planning into the equation." The CIO agreed that the managers needed to get more involved in the process of defining development methods. And while he did not say anything, the CIO thought to himself that giving the users what they wanted was not entirely bad.

Time: Six Years Ago

Shortly after the Director of Applications Development was named Chief Information Officer, he put together a Technology Task Force of senior managers and developers to review developments in information technology and development methods, and report back on the trends in the industry that DSM should be aware of.

The group started by surveying new technology and reading research reports on new development methods. While the work was exciting, it did not generate applicable results. It was interesting to read what the authors of new books were saying, but the members of the task force often found it difficult to translate trends into action items for DSM. The task force continued its work over the years. Every six months they issued a report.

There were successes. The work of the task force was instrumental in introducing Java to the development environment. They also got credit for aiding in the introduction of Web-based applications into the DSM framework.

Along with their search for trends in technology innovation, the task force worked with the SQA analysts to identify and define standards to be used by the developers while building work products. For example, along with the

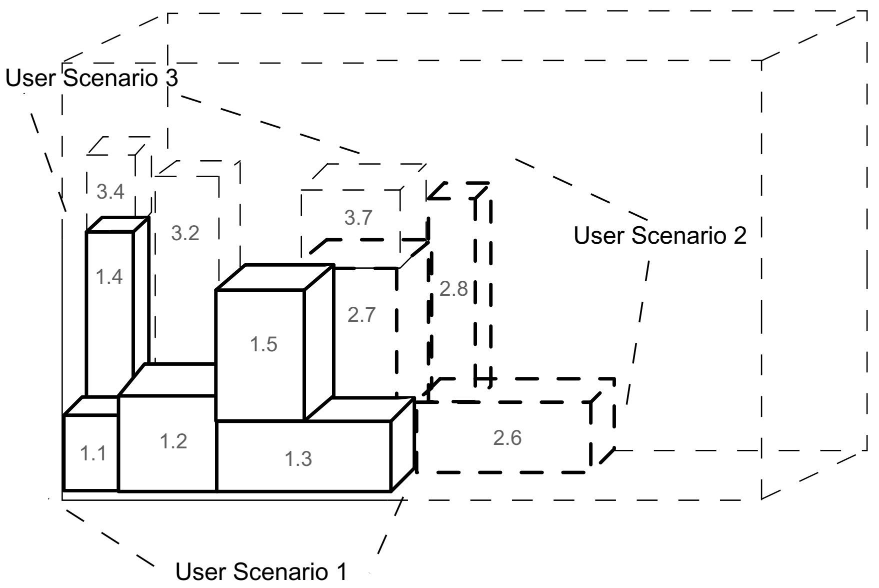


Exhibit 32 Methodology as Continuous Build (a.k.a. Chunk Model)

recommendation of the C++ development environment, the task force searched the literature for guidelines and best practices — including coding style guidelines and class format templates. When the task force pushed for adoption of object-oriented design, they included a proposed standard for the style and construction techniques to be used (these were based on the approach advocated by the consulting firm selected to conduct training seminars).

The task force was very active in the definition of the project repository. Its members energetically participated in the online discussion and worked with the developers on the definition of the database prototype. They tried to find ways to add their standards and framework definitions to the idea of a development repository. Several of the task force members were at the famous lunch session when the manager from Midwest drew the "chunk" model (see Exhibit 32) on the whiteboard.

The blocks on the diagram became a focus for the task force for weeks after the developers' conference. The members began to talk of the blocks as mini-applications, each being implemented using a known set of technologies following a known set of standards.

The task force began using the vocabulary that was developing among the developers. Small clusters of these mini-applications were described by user "scenarios."

They decided there was another dimension to this model. Each of the blocks is associated with a set of default design decisions. If the block was part of the Web-based systems, the developers would use the current Web server and Web scripts. If the block included access to internal data sources, the developers would follow the audit requirements defined in DSM data administration standards.

Exhibit 33 Proposed Element Definitions with Framework

Element Name	*Purpose*	*Framework*	*Standard*
Requirement	Description of an application's properties and behavior expected by a user	Use Case in T-Soft	DSM-422
Object Design	Model of a class highlighting public methods and interfaces	Class Diagram in T-Soft	DSM-31
Relational Table	Physical structure in a relational database	Table Design in SQL Server	DSM-19
Test Case	Definition of initial condition, input, and expected result used to verify an application	Boundary Analysis in Word document	DSM-485
C++ Class	Unit of software compiled with V-Studio used for our client/ server applications	Compiler Y version 2.1	DSM-60

When the developers outlined the idea of elements and element definitions, the task force proposed that each element be associated with some component of the framework (see Exhibit 33).

Some of the database designers reviewed the idea of adding the framework and standards attributes to the element table and concluded that a better design would be to treat the items of the framework and the standards as elements in their own right. Each item of technology (such as Boundary Analysis and Use Cases) and each standard (e.g., DSM-422 defining the qualities of Use Cases, and DSM-485 defining the characteristics of boundary analysis test cases) would be defined as elements.

Association definitions would then record the connection between the elements defined by the developers, the technology used to record/build the elements, and the standards describing desirable characteristics of the elements. A sample of the association definitions is shown in Exhibit 34. The members of the task force found this idea intriguing. One of the managers on the task force commented, "This will make our standards an integral part of the process rather than an afterthought."

Time: Five Years Ago

The push for incremental, repository-based development had many of the DSM managers and the SQA analysts concerned. They expressed their discomfort to the CIO by saying the developers seemed to be acting more like hackers than professional developers. They reported that the code being produced seemed to be of good quality (there was a 64 percent reduction in the number of defects reported in the first six weeks of operations). And they liked the fact that the developers were actively reviewing each other's products. But, there was no formal measurement like they had when they were following the DSM-DM.

Exhibit 34 Examples of Framework Association Definitions

Association →			*← Association*		
Element	*Meaning*	*Avg. Freq.*	*Avg. Freq.*	*Meaning*	*Element*
Requirement	recorded as	1	1	represents	Use Case
Object Design	recorded as	3	15	represents	Class Diagram
Relational Table	implemented as	1	1	implements	SQL Table
Test Case	based on	1	1	used to derive	Boundary Analysis
C++ Class	compiled by	1	n	used to compile	Compiler Y
Use Case	guided by	1	n	std. used for	DSM-422
Class Diagram	guided by	3	n	std. used for	DSM-31
SQL Table	guided by	1	n	std. used for	DSM-19
Boundary Analysis	guided by	1	n	std. used for	DSM-485
C++ Class	guided by	1	n	std. used for	DSM-60

The CIO could see that the managers were unclear about how they fit into the scheme of things. Was this process going to put them out of a job? What was the connection between the work that the developers were doing and the strategic planning process? How should decisions about resource allocation and project prioritization be made?

The managers and the CIO began a concerted effort to rethink how planning and measurement should best be accomplished, given the good work that was coming out of the ranks of the developers. They started by admitting that the accuracy of the data they had been gathering could be better. The CIO asked if there was a way to gather information that did not require the developers to do extra work. He then suggested that the repositories being developed seemed like a gold mine of useful information.

The SQA analysts and manager began paying closer attention to the online conference and the discussion of the project repository. They began to formulate a model that shifted their focus from the individual deliverables produced in processes defined in the methodology to the changing state of the repository itself (see Exhibit 35). They noted that the developer's tasks could potentially add/change/delete many different types of information stored in the repository. For planning purposes, they needed to study the pattern of these changes.

Out of this analysis came a proposal to maintain an "improvement log" — a simple list of the changes made to the repository. The physical form of the repository included the company's configuration management systems, a newly implemented "pending task list," and the directories where the project documents were stored. They augmented the development environment to update the improvement log as the developers checked-in configuration items and made changes to the project directories. This allowed the information to be captured with nearly no extra work required on the part of the developers.

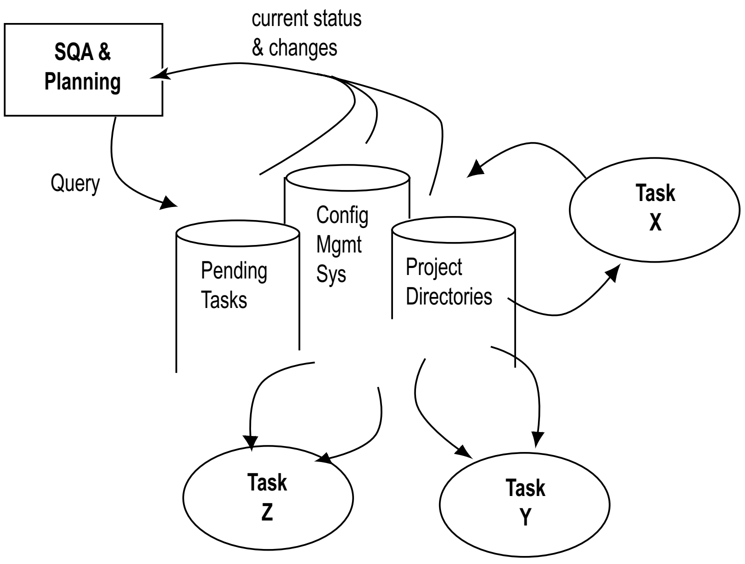


Exhibit 35 Generic Development Process

The SQA analysts began writing scripts to query the configuration management system, the project directories, and entries in the improvement log, and to subsequently produce summaries of activity with flags calling attention to unique situations.

As time went on, the concern and apprehension of the managers and SQA analysts changed to intrigue and surprising enthusiasm. One analyst commented, “We are finally measuring something real rather than regurgitating what others chose to tell us.”

Time: Four Years Ago

While the steering committee did help coordinate the expectations of the users, there were some unanticipated consequences; most significantly, the size of the requests became larger. The steering committee added a layer of formality to the process of selecting projects. For all members of the committee to vote in favor of a proposal, the proposal had to be significant. Smaller projects were viewed as taking valuable resources that should be directed to more “important” work.

The Director and one of his senior analysts were talking one day after a steering committee meeting. The analyst wondered aloud about the trends they had seen in the project approvals. He drew a chart representing the size of a proposed project against the estimated resource requirements.

“You would expect a linear relationship, wouldn’t you?” asked the analyst (see Exhibit 36). “Bigger projects require more resources. But you would expect the approved and funded projects to cluster along a threshold; and proposals perceived by the steering committee as too costly would be rejected.”

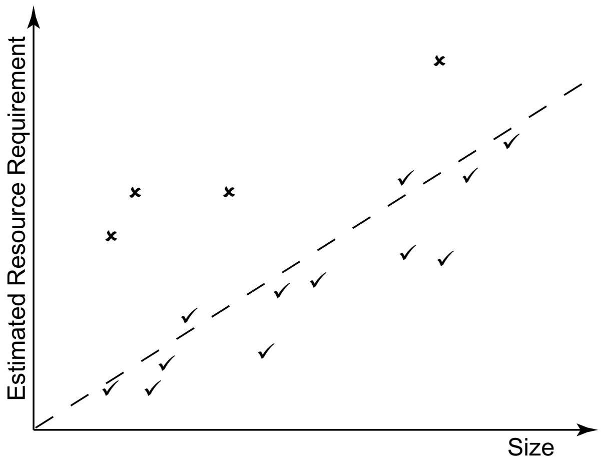


Exhibit 36 Expected Project Funding

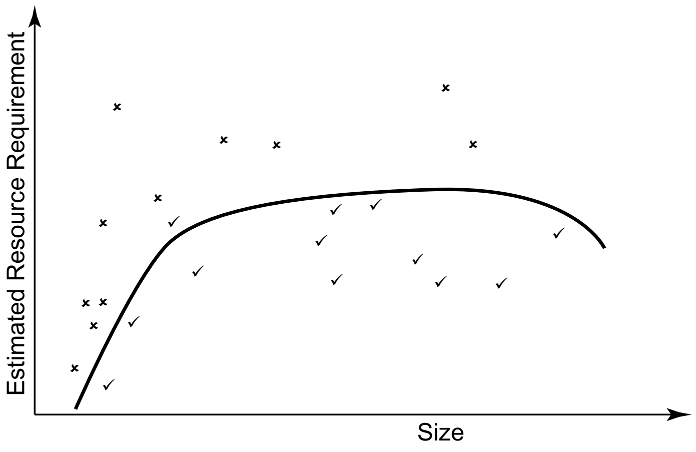


Exhibit 37 Actual Project Funding

"This assumes that perceived benefits are proportional to size," added the Director.

"Right," continued the analyst. "Small project — small benefit. Big projects — big benefits. But that is not what is happening." The analyst started another chart with axes labeled "resources" and "size," but this time he began marking recent project approvals.

The Director and the analyst began listing the proposals recently considered by the steering committee, marking an "x" for rejected proposals and a checkmark for approved proposals, positioning the mark on the graph where it represented their perception of the project's relative size and estimated resource requirement (see Exhibit 37).

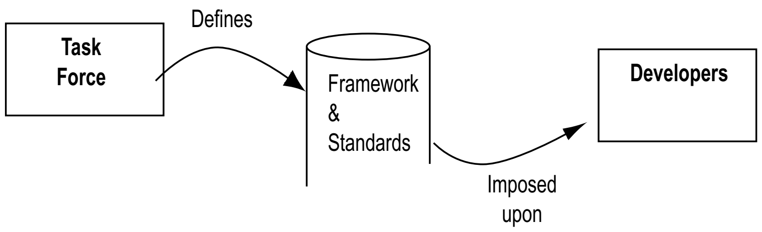


Exhibit 38 Task Force Defines Framework and Standards

- More smaller projects rejected than were larger ones.
- Rejected proposals were the ones with the higher resource estimates.
- As projects got larger, the resource estimates seemed to be proportionally smaller.

"This last point is insidious," commented the Director. "We are under pressure to define proposals that will satisfy the largest number of demands. If we are honest and accurate with estimates, they appear too large. Large projects are nearly impossible to estimate accurately, so there is always pressure to agree to lower resource estimates. And the big projects with lower estimates are the only ones that get approved. Such a deal!"

Time: Two Years Ago

The experiment in incremental development seemed to be working. There was general consensus among managers and developers on the West Coast and at Midwest that focusing on the repository was a viable alternative to focusing on process. People were recording application elements in the repository and they felt good about the accuracy of the information. Recording elements of the development framework in the repository was also working. DSM products were more consistent and innovative because developers were able to work from a template of solutions. Even the idea of defining the developer's goal as enhancing the state of the repository and maintaining internal consistent had become comfortable.

However, another concern developed. When work was performed as a sequence of processes, each process was governed by a set of standards and techniques. The task force had developed these standards and defined them as a fixed part of the process (see Exhibit 38). Developers were not expected (nor did they have any opportunity) to question the standards or contribute to their refinement. The task force would, from time to time, review the standards and make updates, but that refinement was based on task-force findings, and not on the experience of the developers.

Once the items of the framework and the standards were represented as elements in the repository, they became just another part of the knowledge base. The goal is to create consistency. When faced with a conflict between

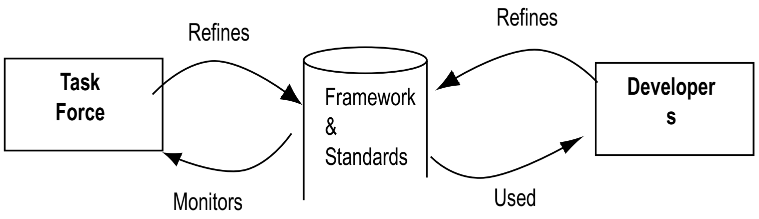


Exhibit 39 Shared Maintenance of Framework and Standards

a design and related code, developers would refine the design or the code or both — choosing the best solution. Now, when faced with a conflict between a design and the design standards, developers might find that the best solution is to change the design or the design standards, or both. The objective is to create an internally consistent set of information about the product. When the developers identified a change to a standard they believed would be constructive, they wanted to be able to change it (see Exhibit 39).

Members of the task force were concerned about their role in the development process and argued that the developers would tend to soften and diminish the standards. The developers argued that their experience in applying the standards yielded positive changes to the standards that deserved to be considered.

For the past year, conflicts between the developers and the members of the task force were few and were handled one at a time with minimal disruption. But finally, a group of developers from the Midwest group proposed a major revision to the code documentation standards and insisted upon documenting their code accordingly. The task force refused to consider the proposal on the grounds that it would result in a different set of documentation than all DSM's older software had used.

The CIO stepped in when he heard the conflict was growing out of proportion. His response was rather unusual. He resisted the temptation to call a meeting with all parties present to "duke it out." He had seen meetings like that end with each side being obliged to defend its initial position even after compelling arguments were presented. The CIO went to the office of each of the task force members and called each developer on the Midwest team. During these private conversations, the CIO asked each individual to explain his proposal and to explain the other side's proposal. He discovered that everyone thought the changes were good and constructive. The only issue from the task force was the fact that existing code and new code would be documented differently.

After the discussions, the CIO met with the task force as a group, summarized the individual conversations, and suggested that the task force modify the current code standards (Exhibit 40), incorporating the suggestions from the Midwest developers. He told task force members that change is inevitable and that they would have to accommodate many generations of techniques,

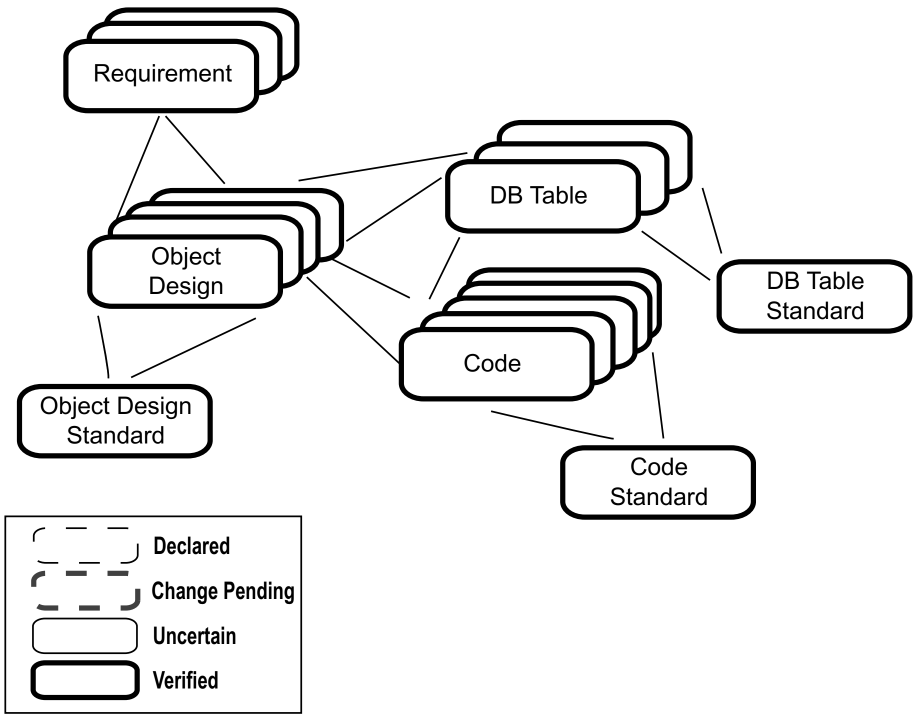


Exhibit 40 Current State

standards, platforms, and product. The CIO then opened up discussion about how to handle the issue of elements being built in different time frames under different standards. This was a safe question because two task force members had outlined their ideas during the one-on-one conversations. For example, one task force member had suggested that nothing be done to the existing code elements. The other member suggested the existing code could be reviewed against the new standard over time.

Approving the new standard changes the state of all associated code elements to “uncertain.” The elements may not need any change but, technically, it is possible that an inconsistency exists between the old code and the new standard that should be reviewed (see Exhibit 41).

No separate development plan would be created. The code would be reviewed only when it was part of another development task. That is, for each future enhancement, developers will be required to review the code in its strand against the new standard. After the code is reviewed and possibly changed, the state of the code element would be changed back to “verified.” Everyone agreed this was a good plan.

The CIO then showed the members of the task force a memo he was sending out that day, stating that suggested changes to DSM techniques and standards were welcome and encouraged from all managers and developers. The recommendations should be directed to the task force. Only the task force had the authority to approve such changes. Before leaving the meeting,

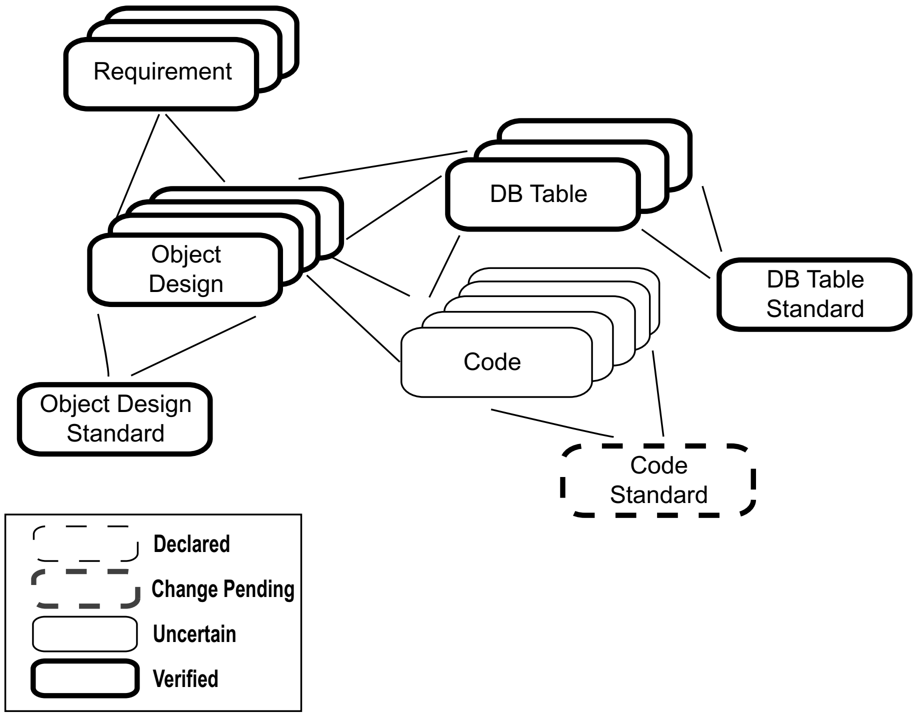


Exhibit 41 Start after New Standard Is Approved

the CIO suggested that task force members should do what each of them felt was appropriate and approve the changes to the coding standards.

Time: One Year Ago

Both developers and users find it easier to maintain a high level of understanding with the advent of incremental development. Most communication occurs on a very informal level with users commenting on the latest change, knowing that their concerns will be addressed in a release in the near future. Both users and developers have reported that the issues being addressed are never big, emotional issues, but rather a series of smaller negotiations (see Exhibit 42).

The development plan intranet site changed from listing projects to listing tasks (see Exhibit 43). The project status pages were replaced with task status pages (see Exhibit 44) with integrated user-to-developer dialogue.

During one discussion, a regional manager recalled how contentious the process was years ago and said, "Now we discuss problems and errors so quickly after they are detected, we don't even call them errors and problems."

The trend toward smaller tasks and funding for continuous development continued at DMS. The use of the online development plan continued to evolve. All parties became comfortable with communicating and refining the direction of the development effort on a continuous basis.

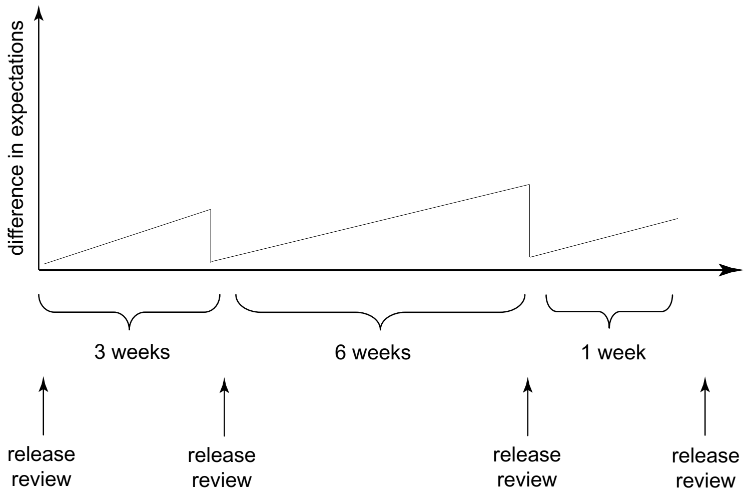
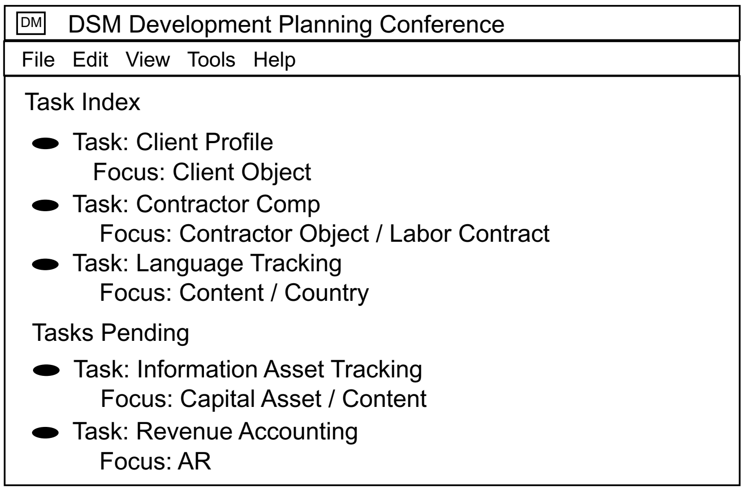


Exhibit 42 Growing Difference in Expectations


DSM Development Planning Conference

File Edit View Tools Help

Task Index

- Task: Client Profile
  Focus: Client Object
- Task: Contractor Comp
  Focus: Contractor Object / Labor Contract
- Task: Language Tracking
  Focus: Content / Country

Tasks Pending

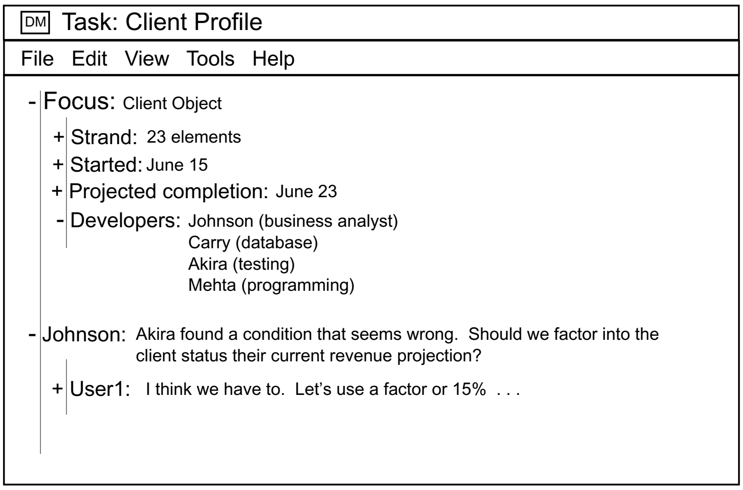
- Task: Information Asset Tracking
  Focus: Capital Asset / Content
- Task: Revenue Accounting
  Focus: AR

Exhibit 43 Current Task List

Conclusion

All organizations are unique, yet all have much in common. I have seen the applications development function of many organizations evolve to follow the patterns described in this case study illustration.

DSM does not exist, but its story is true and it is likely to be similar to your stories. No doubt you recognized situations and problems you faced in your career. We learn by exchanging stories. I am sure as you read the text, you will be reminded of important and illustrative events in your professional life. You are encouraged to share your stories in the discussion at the author's Web site (http://luminguild.com/dynamic).


DM Task: Client Profile

File Edit View Tools Help

- - Focus: Client Object
  - + Strand: 23 elements
  - + Started: June 15
  - + Projected completion: June 23
  - - Developers: Johnson (business analyst)
    Carry (database)
    Akira (testing)
    Mehta (programming)
- - Johnson: Akira found a condition that seems wrong. Should we factor into the client status their current revenue projection?
  - + User1: I think we have to. Let's use a factor or 15% . . .

Exhibit 44 Task Status

Index

A

Accrued work, example of, 33
ActiveX control, 43
Actual cost, 84
Actual work, definition of, 84
Algorithms, decision made by developers about, 92
Applet, 45
Application(s)
  consistent set of elements within, 63
  development
    as capital assets, 141
    department, 199
    reporting of to CEO, 142
  domains, 57, 223
  mini-, 100
  Web-based, 99, 228
Architecture decision spiral, 92
Assembly line, methodology as, 58, 224
Asset development, paying for, 137
Assigned tasks, 138
Association
  cardinality of, 47
  definitions, 44, 45
Assumptions, 9
Avoidance, practicing, 128, 206

B

Best practice(s)
  developer sharing of, 110
  seminars, 110
Bonuses, 106
Boundary Analysis, 101, 230
Browser-based conference, 153
Budget
  decisions, 137
  discussion, 140
  operations, 136
  software development, 162
Business
  analyst, 64, 65
  environment, changes in, 26
  infiltration, 57, 223
Buying decisions, 91

C

C++, 227
  class, 7, 11, 43, 44, 101, 230, 231
  development environment, recommendation of, 99
Capability Maturity Model (CMM), 167–183
Career development, opportunity for, 66
Carnegie Mellon University Software Engineering Institute maturity model, 88, 203
CASE tool, 46
Cash awards, long-term effectiveness of, 70
CEO
  memo to, 141
  reporting of applications development to, 142
Change, 137, 139
  Pending, 81
  risk and, 28
Chief Financial Officer, 95
Chief Information Officer, 56, 94
Chunk(s)
  abstraction of information in, 28
  definition of, 27
  improved versions of, 59
  model, 100
Class
  code, 20
  definition, developer changing, 20
  design, building of, 31
  Models, 15
Clumps, 17
CMM, *see* Capability Maturity Model
COBOL, 44
Code
  associated design, 22
  complexity measurement, 107
  creation of, 19

documentation standards, revision to, 114
elements
built, 25
state of associated, 115
uncertain, 236
error in, 64
information, storage of, 25
lines of, 48
Module, 49, 51
needing review, 39
overly complex, 107
structures, generation of initial, 93
syntax, 31
walk-throughs, 106
Coding
standards, 107, 116
task, ambiguity during, 37
Collaborate
channel, 121, 123, 135
elements, 122
Collaboration, across organizational boundaries, 118
Communication, 117
channels, 118, 119
development group, 127
gap, 104
inter-team, 137
literary, fundamental laws of, 125
managerial, 123
process, improved, 151
Component inventory, 109
Computer Aided Software Engineering tools, 19, 45
Computer-aided system, 12
Computer support, increased requests for, 152
Conference
browser-based, 153
electronic, 148, 149
online, 90, 129
Web, 158
Configuration management, 119
systems, 185, 232
tools, 19
Conflict(s)
-free environment, 111
potential, 111
prioritizing list of, 112
Consensus, formed, 151
Constructive development environment, 103–116
applying dynamic management, 111–112
case study excerpt, 112–116
components of development environment, 105–106
conflict within environment, 103–105
environment assessment, 109–110
ongoing assessment, 110–111
sources of conflict, 106–109
Consulting, 55, 197
Continuous build, methodology as, 58, 100, 225, 229
Continuous development, funding for, 156
Control, lack of, 75
Corporate objective, 119
Cost estimates, 170
Criticism
leadership through, 149
well-intentioned, 67
Culture, creation of, 75
Current task list, 156
Customer
credit query, 52, 78, 138
credit system, 142, 143
database, 138
Object Design, 52
order status, 52, 78, 138
Cynicism, 28

D

Data
definition, 32, 46
element definitions, 32
modeling tool, 46
Database
customer, 138
designers, 230
design tasks, deliverables from, 88
developers working on, 81
management systems, changing, 95
prototype, definition of, 100
schema, 15, 18, 213
system, prototypes of, 74
tables, 34, 59, 73
Day-to-day planning, 41
DBMS
upgrading, 120
users of proposed, 95
vendors, change of, 139
Decision(s)
budgeting, 137
buying, 91
corporate-level, 91
council, common form of, 148
design, 97
dissatisfaction with, 143
implementation, 98
levels of, 95
making, flexible, 147
mapping, 93

patterns, 96
standards, 127
technology, 94
Decision-making process, 94
convenient means of instituting, 147
coordinated, 126
Declared state, element in, 21
Defect
prevention, 182
rates, 104, 109
Delegate
channel, 120, 121, 123, 135
questions, 120
Deposit system, 5
Design
code, ill-described, 26
decision, 97
disagreement involving definition of, 227
elements, written, 25
Developer(s)
alternatives facing, 49
idiosyncracies, 36
inexperienced, 66
less-experienced, 29
memory, knowledge stored in, 24
progress, obstacles to, 105
stereotypes of, 67
UNIX, 199
Windows, 199
workday, 21
Developer management, 61–75
applying dynamic management, 70–71
case study excerpt, 71–75
combining work and learning, 66
developer's view of management, 61–62
diverse skill set, 62–65
motivating by rewarding consistent work, 69–70
multi-team efforts, 67–69
team of one, 66–67
Development
effort
goal of, 63
improvement of, 61
traits of successful, 1
environment(s)
components of, 105, 108
conflicts in, 104
language-based, 106
eras, 57, 223
group, communication within, 127, 221
incremental, 128, 234
management, iterative, 135
manager
key skill of, 2, 3
major duty of, 103
plan, 52, 78
continuous, 170
criticism of, 149
DSM-SISS, 207
misgivings about, 14
staffing requirements, 13, 207
process, generic, 88, 204, 232
projects
errors committed during, 162
long-cycle, 161
repository, 7, 20, 43
schedule, 53
supervisor, 122
unrealistic view of, 4
work dialogue, 36
Director of Applications Development, 143, 152, 217
Discussion(s)
budget, 140
hallway, 16
online, 11, 35, 55, 71, 87, 98, 111, 125, 140, 151
philosophical, 16
proposal, 155, 220
Dissonance list, 110
Diz-dumb, DSM-DM referred to as, 15, 209
Documentation
code, 114
comments casting aspersions on, 62
old, 46
user, 164
Document repository, 185
Dream interpretation, 6
DSM
application development office layout, 200
case study, 197–239
communication across, 126, 221
development
early years of, 202
methodology, 213
framework, introduction of Web-based applications into, 228
older software, 235
product offering, 210
use of technology within, 151
DSM Development Method (DSM-DM), 12, 15, 128, 202, 203
DSM-DM, *see* DSM Development Method
DSM-SISS, *see* DSM Strategic Integration Support System
DSM Strategic Integration Support System (DSM-SISS), 12, 13, 206

development plan, 207
project
disaster of, 55
planning of, 89
requirements statement of, 89
Dual management systems, 157
Dynamic management
getting started with, 163–165
defining building blocks, 164
defining repository, 164–165
measuring productivity, 165
orchestrating pilot projects, 164
selecting comrades, 163–164
what can be lost, 165
information model, 185–188
Dynamic System Development Method, 106

E

Education, enhancing, 108
Electronic conference, structure of, 148, 149
Element
allocating time to, 86
artificially separated associations between, 26
association, 44, 47
and association model, 45
change of existing, 77
code, state of associated, 115
collaborate, 122
complexity of, 33
in declared state, 21
declaring all instances of, 50
definitions
comparison of model with, 98
examples of, 43
expanded, 96
with framework, 101
incomplete, 49
information shared in, 120
listing of transitions made to, 80
mapping architecture to, 96
movement of from declared to uncertain, 77
need to change, 70
new strand of, 27
relationship between, 187
requirements, 63, 82
shared, 119
state model, 29, 34, 68
strand of, 23
tools missing valuables, 46
types, 25, 42
uncertain, 22
unconnected, 50
in Verified state, 69
E-mail
boss's, 56
calls for strategic decision participation sent out by, 151
development staff notified by, 37
director expressing concern, 14
Midwest Developer, 222
review meeting, 15, 209
senior analyst, 207, 208
time reports, 38
Enhanced value
demanding, 84
selling, 123
Enhancement project, 161
Enterprise
creating knowledge about business, 4
management, demand from, 26
managers in user departments of, 124
operation, product's contribution to, 8
structure, 117
value added to, 131
Entity
definition, 11
design, 45
ERP implementation projects, 57
Erroneous truism, 91
Error, potential, 111
Event definitions, 15, 51, 74
Expectations, growing difference in, 127, 128, 206, 238
Extreme Programming, 157

F

Failure, risk of, 133
Formats, decision made by developers about, 92
Framework association definitions, 101, 231
Functions, cluster of, 17
Funding and economic return, 131–145
applying dynamic management, 139–140
case study excerpt, 140–145
funding of perpetual effort, 135–137
funding as risk containment, 131–135
paying for asset development, 137–139

G

Glossary, 189–193
Goal(s)
progress toward, 62
redefining, 4
Goal defining, 1–18
applying dynamic management, 10–12
case study excerpt, 12–18

information as manager's tool, 6–8
element's state, 7
product's state, 7
task's state, 7
knowledge management, 4–5
manager's nightmare, 6
skills and success, 1–4
trusting what you know, 8–10

H

Hallway discussions, 16
Hours
allocation, 85
/element summary, 85
Human effort, risk versus big, 132

I

Ideal, vision of, 1
Implementation decisions, standard, 98
Improvement log, 81, 82, 83, 86, 90, 231
Inconsistency, resolving, 29
Industry
-sector research reports, 198
trade journals, 54
Information
abstraction of in chunks, 28
accessible, 33
accuracy of, 112
assets, lost, 161
-based management, 71, 87
chunk of, 26
code, storage of, 25
development effort based upon changes in, 10
elements, 7
identifying associations between pieces of, 10
as liability, 33
losing track of, 5
as manager's tool, 6
model, 72, 185, 186, 226
need for, 93
organization of in chunks, 27
process of defining, 117
process of gathering, 5
relationships, with company officers, 126, 220
shared, 119
technology, review of developments in, 99
timesheet, 38
visualizing of, 41
Integrated software management, 180
Inter-component influences, 107
Interface(s)
change of, 26
decision made by developers about, 92
designs, 73
Intergroup coordination, 181
Intranet, 129
Inventory
pricing, 68
valuation method, 52, 78, 138

J

Java, 227
applet, 7
application, building of, 32
class, 42, 43, 45, 64, 79, 96
new, 47
syntax for, 31
Development Kit, 109
environment, 94
library, 50
style guide, 109

K

Knowledge
-creating endeavor, 161
management, exercise in, 4
organization of, 27
repository
development of, 3
worksheet for, 11
storage of in developer's memory, 24

L

Lack of control, 75
Language(s), 97
-based development environments, 106
development, 109
third-generation, 200
Leadership by consensus, 147–156
applying dynamic management, 151
case study excerpt, 151–156
decision councils, 147–149
leadership through criticism, 149–150
need for responsibility, 150
Learning
opportunity for, 66
work and, 66
Legacy system
code used in, 22
set of functions from, 7
Liability, information as, 33
Lines of code, 48

Literary communication, fundamental laws of, 125
Loop construct, writing of, 92

M

Maintenance
  activities, developer view of, 62
  phased, 159
  philosophy, changing of, 163
  requests, 143
Manager
  communication, enhanced, 123
  gauges, 9
  nightmare, 6
  relationships, 118
Managing managers, 117–129
  applying dynamic management, 124–125
  case study excerpt, 126–129
  collaborate channel, 121
  dealing with expectations, 117–120
  delegate channel, 120–121
  selling enhanced value, 123–124
  service channel, 121–123
Marketing, proposal to enhance, 148
Meeting(s)
  calling first, 217
  post-mortem, 89, 214
  steering committee, 155, 220
Methodology
  assembly line, 224
  blocks, 212
  continuous build, 225
  definition, 57
  representation, 39, 213
  steps, 211
  use, strengthening of, 108
Midwest Development work area, 227
Mini-applications, 100
Misfit, 103
Misinformation, 51
Model
  Capability Maturity, 167
  chunk, 100
  Class, 15
  comparison of with element definitions, 98
  element and association, 45
  Element State, 29, 34, 68
  information, 72, 185, 186, 226
  maturity, Carnegie Mellon University Software Engineering Institute, 88
  repository, 227
    definitions added to, 74
    enhancement of, 73
  seven-component, 111
  standard implementation decisions, 98
  state, 44
Multidisciplinary teams, 67

O

Object
  Definitions, 51, 59
  Design, 11, 45, 64, 78, 96, 101
    change to, 70
    newly refined, 69
Object-oriented analysis workshop, 109
Object-oriented design, 94, 95, 99, 229
One-person teams, effectiveness of, 67
Online conference, 90
Online discussion
  conference, 129
  index, 11, 35, 55, 71, 87, 98, 111, 125, 140, 151
Operations budget, 136
Order entry sequence, 138
Organization
  chart, 124
    old, 198
    recent, 200
  operation of at disadvantage, 62
  philosophy of, 157
  process
    definition, 180
    focus, 180

P

Paradoxical environment, effective management of, 5
Patterns, conflicting, 104
Peer reviews, 106, 181
Pending tasks, 90, 138, 139, 231
Perpetual effort, funding of, 135
Peter Principle, 3
Phased management, 159–162
  distortion of real objective, 160–161
  long lead-times, 159–160
  lost information assets, 161–162
  specialization of skills, 160
Philosophical discussions, 16
Philosophy, changing of management, 163
Pilot effort, successful, 165
Planning dialogue, 42
Post-mortem meetings, 89, 214
Pricing Function, 21, 22
  code, 23, 29
  enhancement of, 70
Process change management, 182

Product(s)
  attractive interface of, 2
  building blocks of generic, 44
  class design, 50
  cohesive chunks of, 73
  current understanding of, 128
  data definition, 32, 50
  development, 19
  durability, test of, 2
  engineering, 181
  inquiry, 52, 138
    function, enhancement to, 49
    requirement, 50
  inspections, 106
  introducing bugs into, 47
  less accurate information about, 20
  Object Design, 52
  pricing
    formula, 52, 78, 83, 138
    function, 150
  software, chunks of information in, 19
  work, way of querying, 35
Productivity, 54
  defined, 37
  dialogue, 87
  measure of, 165
  monitoring, 77–90
    applying dynamic management, 86–87
    case study excerpt, 87–90
    demanding enhanced value, 84–86
    determining work's cost, 84
    measuring what has changed, 80–84
    measuring work done, 77–80
  numbers, dropping, 86
Programmer, 64, 65
Progress planning, 41–59
  applying dynamic management, 53–54
  case study excerpt, 54–59
  information structure, 41–42
  real building blocks, 42–49
  real uncertainty, 50–51
  rewarding complete thinking, 49–50
  watching result, 51–53
Project(s)
  commitments, 172
  defined software processes, 183
  development, errors committed during, 162
  DSM-SISS, disaster of, 55
  enhancement, 161
  ERP implementation, 57
  failure, 89
  funding
    actual, 145, 233
    expected, 144, 233
  orchestrating pilot, 164
  overhead consumed by, 15
  overview, 38
  partitioning of into phases, 161
  proposal team, 170
  rejected, 145
  requests, 217
  schedule, computed, 37
  software activities, responsibility for, 171
  status
    page, 155, 219
    reports, 38
  tracking, 171, 173
Promotions, 106
Proposal(s)
  discussion area, 155, 220
  rejected, 145, 234
  significant, 144
Prototyping approaches, older, 134

Q

Quality, 86, 88
Quantitative process management, 181

R

Rationalization, 51
Rational Unified Process, 106
Reading list, 195–196
Redundancy, 117
Relational database table, 119
Relational Tables, 78, 79, 96, 101, 230
Reporting relationship, change in, 154, 218
Repository
  adding value to, 29
  changes, 87
    binary, 82
    over time, 10
  contents of, 65
  definitions, 30, 163, 164
  development, 7, 20, 43
  document, 185
  elements in, 9
  gauges in development of, 8
  information in, 162
  knowledge, 11
  model, 227
    definitions added to, 74
    enhancement of, 73
  organized by element type, 25
  organized by individual, 25
  shared, 154, 169, 175, 219
  in steady state, 24
  uncertain elements, 22

Requests
  new development, 216
  prioritization of, 129
  project, 217
  significant, 153, 217
Requirement
  associations to, 47
  definition of, 32
  documents, 14
  elements, 63, 82
  management, 168
  risk versus, 133
  statements, ambiguity in, 205
Resource(s)
  allocation, mismatched, 118
  redirecting, 4
Return on investment, maximizing, 3
Reviewing without assent, 34
Review meeting, 15, 18
Risk
  big human effort versus, 132
  big requirements versus, 133
  big technology change versus, 133
  big timelines versus, 132
  change and, 28
  containment, funding as, 131
  curve, 133
  increased, 134
  small, 136

S

SCCB, *see* Software configuration control board
SCM, *see* Software configuration management
Self-directed teams, 67
Service
  channel, 119, 121, 124
  funding versus, 136
  questions, 122
Seven-component model, 111
Shared elements, 119
Shared information, 119
Shared project repository, 154
Shared repository, 169, 175, 219
Significant requests, 153, 217
Silos, separating of skills into, 160
Skill(s)
  set, diverse, 62
  specialization, 159, 160
Software
  baseline audits, 179
  configuration control board (SCCB), 178
  configuration management (SCM), 175, 178
  designer, 64, 65
  developer work, measure of, 161
  development
    budget, 162
    destructive processes, 131
    goal of, 28
    industry, points of dissonance in, 109
    managers, debate of, 48
    state of, 211
  engineering
    activities, 173
    tools, computer-aided, 19
  estimates, documented, 169
  industry, early days of, 160
  innovations in, 97
  life cycle, 170
  management, relating CMM to dynamic, 167
  managers, first-line, 172
  products, chunks of information in, 19
  project
    commitments, 172
    planning, 169
    tracking, 171, 173
  quality assurance (SQA), 89, 169, 176, 214
    activities, funding for, 176
    analysts, apprehension of, 90
    group audits, 177
  quality management, 182
  requirement, 28
  solution(s)
    building effective, 70
    effective use of, 2
  subcontract management, 174
  subcontractor statement of work, 175
SQA, *see* Software quality assurance
SQL
  Server, 230
  Table, 101
Staffing requirements, 13, 207, 208
Standard(s)
  ambiguous, 108
  approval, 115, 237
  coding, changes to, 116
  decisions about, 127
  lack of, 108
  verbiage, 106
State
  current, 114, 236
  models, 44
Statement of work, software subcontractor, 175
Steady state, initial state to new, 63
Steering committee
  creation of, 216
  meetings, cancellation of, 155, 220
Strand, unknowns in, 32
Strategic framework, 91–101
  applying dynamic management, 98

architecture's dark side, 97–98
case study excerpt, 99–101
importance of system architecture, 91–94
mapping architecture to elements, 96–97
technology decisions, 94–96
Styles, decision made by developers about, 92
System
architecture, importance of, 91
definition of, 215

T

Talent, capitalizing on, 3
Task(s)
appropriate, 215
approved, 150
assigned, 138
change, development steps taken during, 29
coding, ambiguity during, 37
definition, 19, 28, 30, 69, 150
estimated work requirements for, 79
force, 100, 101
defining framework and standards, 234
framework and standards defined by, 113
updates made by, 113
list, current, 156, 238
pending, 138, 139
size, factors determining, 33
state, 8
status, 156, 239
strand, 30
team assigned to, 64
unrealistic, 8
Team(s)
effectiveness, 112
multidisciplinary, 67
one-person, effectiveness of, 67
problematic, 66
progress, measurement of, 35
project proposal, 170
responsibilities, 65
selection, 163
self-directed, 67
shared element as bridge between, 69
Technical reviews, 175
Technology
change
management, 182
risk versus big, 133
decisions, 94
two-generations-old, 134
use of within DSM, 151
Test
cases, boundary analysis, 230
scenarios, 59
suite, 42, 43, 63, 79, 96
completed, 83
responsibility for, 65
Tester, 64, 65
Third-generation languages, 200
Timelines, risk versus big, 132
Time report(s), 39
e-mail, 38
memo on, 205
sample, 37, 204
Timesheets, information on, 38
Training
/education, definition of, 106
program, 180
Two-generations-old technology, 134

U

Uncertainty, 50
Units of measure, predictive, 48
UNIX developers, 199
Use cases, 74
User
-to-developer dialogue, 237
documentation, 164
manuals, 72
needs
data supporting, 187
financial constraints at expense of, 161
requirements, understanding of, 59

V

Value, evidence of, 131
Vendor
DBMS, 139
representatives, 95
training, experiences with, 110
Verification activity, 34
Vocabulary, 228
VP of marketing, frustration from 201

W

War room, 40, 74, 214
Web
conference, 158
sites, developer-oriented, 84
Web-based applications, introduction of into DSM framework, 99, 228
Web-based systems, 100
Windows developers, 199

Work
  accrued, 33
  backlog of, 153
  cost, determining, 84
  defining, 19–40, 67
    applying dynamic management, 34–35
    case study excerpt, 36–40
    defining tasks, 28–30
    developer's workday, 21–23
    projecting size, 30–33
    relating management to work, 24–26
    unit of work, 26–28
    work and corroboration, 33–34
  development, dialogue, 36
  disrupting productive, 165
  error-prone, 51
  learning and, 66
  measure of software developer's, 161
  measuring amount of, 77
  motivating by rewarding consistent, 69
  need to enhance original, 18
  products, way of querying, 35
  record, 53
  relating management to, 24
  reuse of existing, 94
  sequence of processes, 112
  use of communication channels to encourage creative, 123
  use of plan to predict, 48
Workshops, impromptu, 56, 222

X

XYZ enhancement, 86

INFORMATION TECHNOLOGY AND PROGRAMMING

Dynamic Software Development

Timothy D. Wells

Dynamic Software Development: Managing Projects in Flux defines the principles, practices, skills, and techniques needed to manage a dynamic software development environment. At a hands-on level, the book helps managers define the project goal and the actual situation, plan progress, manage developers, and monitor productivity. At a higher level, the book helps managers determine a strategic framework, ease workflow in the development environment, obtain funding, increase economic return, and implement leadership by consensus.

Targeted at those who manage information systems, corporate information, and developers, the book features a section at the end of each chapter to help them apply and customize the recommended techniques to their specific organizations. This reference addresses recent approaches to building applications such as Extreme Programming, Adaptive Software Development, and "lightweight" methodologies. By noting the failure of similar techniques in the past, the author emphasizes that such ideas can only achieve their true potential via the common, consistent management techniques outlined in ***Dynamic Software Development***.

Features

- Offers a detailed, specific guide for managing the entire development effort, including new development and software maintenance
- Supplies a strategy for managers to effectively control the development effort without imposing artificial and burdensome constraints on the developer
- Illustrates how to customize and apply the techniques provided to each specific organization
- Includes effective, non-linear management techniques that can respond to the intensifying demand for Web-based applications, and the growing pressure on development teams to improve their offering

About the Author:

Timothy D. Wells is a recognized expert in project management and project planning. Mr. Wells is an associate professor in Information Technology at the Rochester Institute of Technology. He has more than 28 years of experience in the software industry. His current focus is on information asset management and the effective use of technology for improving organizational performance.

AU1292

ISBN 0-8493-1292-2

90000

9 780849 312922

AUERBACH PUBLICATIONS

www.auerbach-publications.com