Mastering Software Project Management

Managing Software Projects According to The Art Of Business

copyright by Hsiang Tao Yeh

December 2006
First Edition (1.2)
Published by www.lulu.com

Other books by the author:

Five Willows Guy
Follow Your Blessings
Fragrant Orchids of Hidden Valley
Bodhi Tree
Bai Hua Ru Shi (with Nian Qing)
Sunset Is Still Calling

My e-books could be downloaded for free at
http://people.lulu.com/users/index.php?fHomepage=101324

Dedication

This book is dedicated with love, gratitude, and admiration to
My dear brother Dr. Raymond Yeh and
My dear sister-in-law Priscilla Chow

Preface

Even though software engineering is formalized into a discipline not quite fifty years ago, software applications today are pervasive and critically important in many areas of society. Although there are many good books about project management and software management, the focus of existing books seem to be either on detail mechanics and tools of project monitoring and task tracking, or about high level ideas and approaches on how organization's capabilities might be improved. There seems to be little offered to address various specific issues in an individual project to build long-lasting repeatable success.

I have been very fortunate to have opportunities to work on software quality and software development in AT&T over the last twenty some years. Over the last dozen years, I had management responsibility as development manager for various projects and learned from experiences a number of techniques that worked well for me. As a post-retirement project, I decided to collect my ideas, experiences, and some of the techniques I found useful in software project management into a small book, in the hope that what I offered here will be useful to others. Software project manage-

ment could be very challenging at times, but can also be very much fun and rewarding. I hope this book can help future software project managers not only to be successful in their projects but also to enjoy the assignment as well.

This book project got a big boost through recent collaboration with my dear brother Dr. Raymond Yeh. Ray is a pioneer in software engineering and made many fundamental contribution to the field. He is also a natural in the art of possibility, never seems constrained or limited in any way by his position or training, and is always ready to flow with whatever opportunities presented to him in life. He had always been interested in management and management philosophies. Recently, he and my dear niece Stephanie wrote a ground-breaking book, "The Art Of Business - In The Footsteps Of Giants".[1] He also invited me to join him in giving joint seminars on Technical Management as part of Texas Tech University's special training program. In this way, I got to try out the materials here and got some good feedback. In this regard, I am also most grateful to Prof. Atila Ertas of Texas Tech University to include me as part of his team in giving the training classes over the past couple years.

While I always benefited in the past from discussion with Ray on various management issues, his new book provided a unifying framework that my experiences in software project management fit in very naturally. The need to give seminars also provided the necessary impetus for me to put my ideas on software project management down

[1] web site of the book at http://theartofbusinessbook.com

on paper, first as viewgraphs and draft notes, and then this book.

This book is organized following closely the sequence of the five arts of business given in Ray and Stephanie's book. These five arts are, the art of possibility (vision), the art of timing (logistics), the art of leverage (leverage), the art of mastery (process), and the art of leadership (teaming). These arts are very general management principles and can be applied to many situations. In my book, "Follow Your Blessings", I suggested ways to apply these principles to manage personal life and relationship. Here, a uniform organization format will be used in the following chapters to discuss each art of business, including: provocative statements, key ideas and techniques, useful practices, examples and stories, and how to evaluate a project. The focus will be on technical problems that one encounters frequently in software project management, rather than on the mechanics of project monitoring.

The book also contains some new approaches and insights that have not been published before. Foremost is the application of the five arts of business to project management. As an example, let us consider the many leverages one could deploy in the architecture of a software system - from component-based platform, review of key architectural issues by experienced architects, pattern-based architecture framework, and domain specific scripting language. By following the concept of "market inflection point" of the art of leverage, and knowing the many leverages one could apply in parallel, one has a simple recipe on hand to elevate the architecture maturity of a software

system to a very high level. In this book, I also have specific suggestions and examples on how to go about doing statistical process control for software processes, a topic of a certain amount of current interest.

I have learned from many throughout my career at AT&T. Unfortunately, AT&T today is a very different company from the one I joined back in 1979. But I am still very grateful to AT&T for the opportunity to learn about software programming, process quality and project management. The work environment and people were excellent and it had been an excellent work experience for me. I am also most grateful to my many colleagues, managers, peers and project team members. Without their help and support, not only I won't learn anything, but the projects also won't be successful. Many have directly influenced me on how to go about software project management or process quality management, including: Dave Chai, Erold Hinds, Steven Blazier, Helen Hwang, Bruce Gundaker, Hosein Fallah, Dick Hawkins, K-C Li, Shuhshen Pan, Peter Ng, Yee Lee, Peter Ting, James Chang, Pat Reilly, J-T Hsieh, Bill Weinberger, Jar Wu and others. I am most grateful to their input, example, and help.

My two dear brothers, Dr. Raymond Yeh and Dr. Randy Yeh, have been my constant guide and coach for my entire career at AT&T. They have always given me encouragement, great advice, and much help in many ways. Randy, in particular, convinced me to switch from physics to software. What a great decision that was for me. From time to time Ray shared with me generously his quest and insight for holistic manage-

ment framework. His thinking invariably robbed off on me and helped me to be able to connect my learning in software project management into consistent and organic pattern that seems useful and functional.

I dedicate this book to Ray and my dear sister-in-law Priscilla Chow, who together with Ray have helped me, my immediate family and the whole Yeh clan in numerous ways over the past thirty plus years. I am deeply greatful and feel very blessed and lucky indeed to have so wonderful a pair as my close relatives.

Note for edition 1.2: A new chapter "Lessons From Stories" has been added.

Contents

1 Introduction **14**

1.1 What This Book Is About? 14

1.2 Keys to Better Project Management 16

1.3 Traditional Project Management . 17

1.3.1 Some Typical Activities in Project Management[FAR] 17

1.3.2 Some Approaches on Project Planning 18

1.4 Project Management is Much More than Task Monitoring 19

1.5 Manager as Problem Solver 21

1.6 Some Tough Technical Management Challenges 23

1.7 Management Framework Changes from Year to Year [MCG] 25

1.8 Our Approach 26

2 Vision **28**

2.1 Statements about Vision 28

2.2 Vision as A Base for Other Activities 29

2.3 CMM Roadmap and Framework . . 31

2.4 Using Vision to Motivate 32

2.5 Some Project Vision Examples . . 33

2.6 Stories about Vision 33

2.6.1 Clarifying Vision for a Project 33
2.6.2 Don't Forget the Big Picture [CRA] 35
2.7 Some Useful Practices in Vision . . 36
2.8 Sharing and Project Evaluation . . 37
2.8.1 Sharing about Vision 37
2.8.2 Evaluation of Project Vision 38

3 Timing and Logistics **39**
3.1 Statements about Project Timing Issues and Logistics 39
3.2 Some Software Timing Issues and Logistics Challenges 41
3.3 Anticipate and Prepare for Risks, Changes, and Crisis 43
3.4 Could Configuration Be Unified or Simplified? 46
3.5 Examples and Stories 47
3.5.1 Some Stories on Risk Management 47
3.5.2 Case Study on Risk Aversion 48
3.6 Useful Practices in Timing and Logistics 49
3.7 Sharing and Project Evaluation . . 50
3.7.1 Sharing in Timing Issues and Logistics 50
3.7.2 Evaluate Risk/Logistics Management in Projects . . 51

4 Leveraging Software Manufacturing Infrastructure **52**
4.1 Leverage Comes in Many Forms . . 53

4.2 Statements About Leveraging Software Manufacturing Infrastructure 54
4.3 Build/Assemble Your Own Software Factory 55
4.4 Leverage Open Source Software . . 56
4.4.1 The Advantage of Open Source 56
4.4.2 ANT - Another Neat Tool [HAT] 59
4.5 Leverage Organizational Resources 60
4.6 Stories in Leveraging Software Environment and Components 61
4.7 Useful Practices in Leveraging Software Environment and Components 63
4.8 Sharing and Project Evaluation . . 64
4.8.1 Sharing on Software Environment 64
4.8.2 Evaluation of Software Environment 65

5 Leveraging System Architecture Framework 66
5.1 Statements Related to Leveraging Architecture Framework 67
5.2 Some Architecture Concerns and Design Principles [FOW, VAS] . . . 68
5.2.1 Key Architecture Issues . . 68
5.2.2 Enhancing Performance and Capacity 69
5.2.3 Enhancing Reliability, Availability and Flexibility . 74
5.2.4 More Ideas and Solutions on Architecture Issues . . . 76

5.3 Architecture Patterns and Styles . 78
5.4 Leverage Architecture Review . . . 80
5.5 Software Engineering with Component Rich Java 2 Platform 83
5.5.1 Leverage Technology - Reusable Components, Design Patterns and Architecture Framework 83
5.5.2 Java 2 Platform 84
5.6 The Rise of Design Patterns in Software Engineering 86
5.7 Assemble Design Patterns into Architecture Framework 89
5.8 Could One Work at a Higher Level? - Problem Domain Scripting 91
5.9 Stories in Leveraging Architecture Framework 93
5.10 Useful Practices in Leveraging System Architecture Framework 94
5.11 Sharing and Project Evaluation . . 96
5.11.1 Sharing in System Architecture Framework 96
5.11.2 Evaluation of System Architecture Framework 97

6 Process Discipline 98
6.1 Statements about Process Discipline 99
6.2 Some Process Examples 101
6.2.1 Rational (now IBM) Unified Process (RUP) [RAT] . 101
6.2.2 eXtreme Programming(XP) [EXT], Is It for You? 102
6.2.3 Some CMM Key Process Areas 103

6.3 How to Improve Process Quality . 104
6.3.1 A Six-part Framework [YEH] to Understand a Software Process 104
6.3.2 What Is Quality? 105
6.3.3 Some Data Analysis Techniques Applicable to Software Processes 106
6.4 Statistical Process Control for Software Processes 109
6.4.1 Control Chart Concepts [GRA-2] 109
6.4.2 Six Sigma Concept [SIX] . . 113
6.4.3 Can One Apply Process Control Charts to Software Manufacturing? 114
6.5 Modeling in Software and Statistical Process Control 116
6.5.1 A Simple Software Reliability Model 116
6.5.2 Lognormal Distribution for Data from Groups of Projects 118
6.5.3 Model for Code Inspection Effectiveness [CHR] 120
6.6 Stories about Process Discipline . . 121
6.7 Useful Practices in Process Discipline 122
6.8 Sharing and Project Evaluation . . 124
6.8.1 Sharing about Process . . . 124
6.8.2 Process Evaluation 125

7 People Development and Team Building 127
7.1 Some Statements Related to People Development and Team Building 127
7.2 People Motivation 132

7.3 Trust and Relationship Building [FIS] 134
7.4 Approach to Negotiation [FIS-2] . . 136
7.5 Ideas on Team Building 136
7.6 A Story about Self-Managed Team [YEH-2] 138
7.7 Useful Practices in People Development and Team Building 140
7.8 Sharing and Project Evaluation . . 142
7.8.1 Sharing about People and Team Development 142
7.8.2 Evaluation of People Harmony in Projects 143
7.9 Technical Management - Opportunities for Action 144

8 Lessons From Stories **147**
8.1 Don't Go Nuclear - Lessons From the Cuban Missile Crisis 147
8.2 The Start of First World War - A Cautionary Tale of Unintended Consequence 153
8.3 Grameen Bank - Lending Money A Little Differently 159
8.4 Ashoka - To Empower Thousands of Social Change Makers 166
8.5 Greg Smith - How To Survive Catastrophe And Live To Tell . . . 173

Chapter 1

Introduction

1.1 What This Book Is About?

The five arts of business management as introduced in "The Art of Business" [YEH-4] is very general and powerful. It is useful to illustrate these management principles for various specific situations. This book is focused on applying these management principles to technical projects, and specifically, software and system project management, where the author has many years of experiences.

To apply the management principles in "The Art of Business" to software and system project management, the five arts of business management will be introduced and discussed in the following six chapters:

- **Chapter 2 - Vision (the art of possibility)**
- **Chapter 3 - Timing and logistics (the art of timing)**

- **Chapter 4 & 5 - Leverage software manufacturing infrastructure and system architecture framework (the art of leverage)**
- **Chapter 6 - Process discipline (the art of mastery)**
- **Chapter 7 - Develop people and build team (the art of leadership)**

A uniform structure will be used in each of following chapters to explore the management principles in each art of business. I shall start by making some thought-provoking statements, then get into key concepts and related management principles. This will be followed by useful techniques and practices, use case examples for illustration. The final section suggests some questions to ask to apply the materials presented in project evaluation and assessment.

Here is some details about my background in software project management. I was trained in physics (Ph.D., Univ. of Illinois), but had been doing software development, software process quality and software project management for over twenty years at AT&T. The projects I worked on are mostly building network management systems, service ordering, and provisioning systems to support telecommunication services. Most of the projects are developed in UNIX/C/C++ environment, and more recently, in Java environment. My project experiences is thus limited to narrow domains and to certain types of projects. However, I think many of the problems project managers need to solve are quite common across industries. I have also worked on

many aspect of software quality and software process control, including AT&T Bell Laboratories wide quality reporting responsibility for projects across all R&D Divisions. I hope the approaches and pointers provided here are useful to the readers.

1.2 Keys to Better Project Management

Another goal of this book is to help the readers to become more effective software or system project managers. The ingredients of better project management include:

- Understand the essential management principles and techniques, such as those covered in "The Art of Business".
- Always learn from one's own project lessons. One should also study and learn from many real-life project case examples. As the number of projects each manager can be personally involved with is usually quite limited, it is imperative to learn from others. There are many good resources in the internet on this. Several such examples will be highlighted in this book.
- Familiarize oneself with all the best current practices in software project management.
- Go beyond just the mechanics of project planning and tracking and learn many powerful ways to improve the project team and the project infrastructure.

- Develop the ability to evaluate a project, to identify its strength as well as its weakness, and to spot opportunities for action.

1.3 Traditional Project Management

Before I get into the more interesting aspects of project management, it is useful to review briefly some of the traditional functions of project management in project planning and monitoring.

1.3.1 Some Typical Activities in Project Management[FAR]

What are some of the typical tasks in traditional project management functions? They are usually grouped in planning, doing, and monitoring tasks:

- Planning - Planning includes such tasks as: to set the overall objectives for the project; to decompose a big task into component tasks; to decide upon the process methodology and lifecycle phases to use to construct the software system; to find people with the right skills and external recruiting; to assign individual project team members with specific tasks and responsibilities; to develop a project plan; to develop a quality plan; to identify logistics needs such as hardware, development environment and tools, long lead time items, etc.

- **Doing - Each project team member starts to perform his or her specific assignments and tasks.**
- **Monitoring and Adjusting - This includes: collection of progress reports, to evaluate the actual as compared with the plan for various metrics and milestones; and to make adjustment and correction as needed**

1.3.2 Some Approaches on Project Planning

How does one estimate how long and how much effort is needed to do a project? There are many techniques for project planning. It is useful to use more than one way to come up with estimates in time, effort, and cost. These estimates can then be used to cross check with each other.

If historical data are available from similar projects, one can apply regression analysis to come up with an estimate for the project. For example, let us assume that function point was used as size metric for past projects. Function points can typically be calculated from details of the system requirement. Regression analysis from data of past projects can provide a relationship between project size (as measured by function point) and project effort (as measured by person-week). From such a formula one can estimate the effort required to do the new project based on the function points calculated from the system requirement.

If there is no historical data from similar projects, one could resort to various estimation

formulas which are derived from data from many different type of projects in the software industry, each formula is a little different in their modeling assumptions about software project parameters.

Another technique in project cost estimation is to do a detailed task decomposition for all system lifecycle phases and solicit input from project team members for time and effort estimates on these tasks. Together with an understanding of task dependence (such as testing tasks should be done after coding tasks of the corresponding modules), one can get an estimate of the overall time and effort. This analysis will also provide an estimate on the staffing profile as well as identifying critical path items for the project. These critical path tasks could not be done in parallel and hence set a limit for the minimal time the project could be finished.

The cost estimates should be revised as project has progressed and real cost from earlier phases are collected. As project moves forward, typically, the project uncertainties are reduced and more accurate estimates for the remaining work can be given.

1.4 Project Management is Much More than Task Monitoring

The mechanics of project management as well as how to use a particular project management tool to monitor tasks in a project are both important. However, they are in my view only

a small part of the true art of project management. There are many good books on these topics. These topics will not be the focus of this book. The questions I try to address in this book is how to have repeatable, successful project management experiences. Also, if one starts out in a weak project team or project environment, how to build it up so that it becomes more and more powerful and successful. The truth is that one could have the best project management tracking techniques and tools and still fail miserably, because why a project succeed or fail has little to do with the tools or techniques used to monitor or track it.

One needs to go far beyond task monitoring in order to become an effective project manager. The reason is that for project to be successful, such that one can deliver products on time or ahead of the schedule, meet high quality standard and make your customers happy, and with tight budget control, many things need to go right every time. Success is not an accident. The project plan may look great on paper, but that alone will not give one confidence to deliver project on schedule, and meet quality, cost and other objectives.

Another way to say the same thing is to ask yourself the following questions. Suppose you do have choices about the following, what would be your answers?

- What people/team you would like to have on your project?
- What processes would you like to use?

- What development environments would you choose?
- What business conditions and corporate culture would you prefer for your projects to be embedded in?

The point is that all these factors would affect how your project might turn out. The art of project management is to set things up so that most of the things would turn out right and helpful to your project. How to get there? By continuous improvement. Project manager's job is never done. It's like building a factory ("software factory"), not just doing a single project. We'll explore many of these questions in the following chapters.

1.5 Manager as Problem Solver

Before we get into the details about project management issues, let's ponder for a minute how the role of project manager or leader differs from that of a technical worker. One aspect of the differences lies in the type of problems encountered. Technical workers may require deep technical knowledge but usually only focus on problems within a narrow domain. In contrast, part of the fun and challenge for a manager is s/he needs to deal with all kinds of problems ("the buck stops here"), frequently making decisions under uncertainties.

So managers should learn and acquire problem solving and decision making skills. There are

many good references in this area [HAY]. Here, I shall just mention a few points:

1. Many management gurus [COV] have stressed the differences between effectiveness versus efficiency. It is important to be efficient. But it is even more important to solve the right kind of problems. So one needs to prioritize, do "first things first", and be clear on your objectives. When I need to make decisions, I try to ask myself - "what problems am I trying to solve?"

2. No need to optimize everything. There's always a limit in a project's capacity - time, resources, energy. For many issues, frequently good enough is really good enough.

3. Balance reason with emotion. In your decision making, you not only want to have the right reasons, you also want the decision to feel right.

Since project managers typically do not work on technical work directly, how can one evaluate their effectiveness? Checking that a project is completed on time, meet budget, or have high quality for a single project may not be an accurate guide. The person could be just lucky. One can get "results" by harsh pressure. As one of my director at AT&T asked us project managers - "What kind of managers are you? Do you take people in, use them up and spit them out? Or you cherish people and help them grow?" There are generals who can win some battles over the dead bodies of many of his/her soldiers and there are

managers who achieve "success" for a moment by slave-driving workers, cut investment for the future or just focus on painting a great appearance. To evaluate whether a manager has really been effective, one needs to look at whether s/he has improved the process, strengthened the people, accumulated technical assets, clarified the vision and seized the opportunities, in additon to delivering project results and consistently exceeded the customers' expectation. Consistent long term success is the key to tell how effective is a project manager.

In fact, the five arts of business management are the excellent natural starting point for the manager with a new project. S/he could evaluate the project from the management principles of the five arts and prioritize what are the opportunities that s/he should focus on. This is what I found very helpful for my own work as software project manager with many projects over the years.

1.6 Some Tough Technical Management Challenges

A recent survey on large public-funded software projects in UK [BCS] found that only 16% were considered successful. In the report, key factors critical to IT project success include: effective project management, risk management, importance of the role of system architect, and professionalism and education for the IT professionals. Other recent surveys [COR] also reported

very high (70%) failure rate in IT projects in general.

In today's business and technical environment, there are plenty tough challenges, including some new ones, for the project managers:

- **How to keep up with and leverage technology to build systems cheaper and faster?**
- **How to automate business processes to reduce cost and add value?**
- **How to motivate people in spite of massive IT outsourcing and lack of job security?**
- **How to compete with software that's free (open source)?**
- **Who to assign "boring" work to?**
- **Is agile eXtreme Programming (XP) the right approach for your project?**
- **How to deal with complex system configuration, multiple versions, and frequent updates?**

We'll address these and many more topics in the chapters that follow and hopefully by the end of this book to have changed these challenges into opportunities for action.

1.7 Management Framework Changes from Year to Year [MCG]

Management framework captures breakthrough ideas in management and there have been many new ideas in management over the years. Different ideas may be emphasized at different time due to perceived needs and sentiment of the society. While it is not quite as variable as fashion that changes from year to year, what's popular or "hot" in management is certainly not standing still. Just recall some of the buzz words of past years such as – management by objectives (MBO); one minute management; management by walking around; downsizing; outsourcing; restructuring; reengineering; skunk works; total quality management (TQM); zero-based cost control; self- management team; automation, etc.

This change in management framework over time should make one skeptical about people who claim that they have found the last words in management. One should check carefully with reality and confirm by comparing with what works for you. However, the fact that no single management framework is popular forever should not detract us from two important points, namely:

1. **Have a management framework is very helpful, much better than ad hoc management. Management framework is like a philosophy of life. It provides a roadmap or general guide to deal with the various problems that can arise in projects. It tells us what's im-**

portant and points us in the right direction.

2. Different management frameworks while emphasize on different things are usually compatible with each other. There are not too many management approaches that believe in just doing one thing and exclude or neglect other things in project.

1.8 Our Approach

While the five arts of business management framework I use here will not be the last word in management, I do believe that it is a comprehensive framework and covers all the important areas that technical project managers need to pay attention to. In my over ten years of experiences as software development and project managers, I have yet to come across problems that do not fall under one of the arts of business for solutions. It is a holistic approach with all the arts supporting each other. When supplemented with detail techniques and practices in each art for the problem domain, I think it is an ideal approach for technical project management.

In the approach here, I try to

- use comprehensive coverage of technical management problem solving principles and techniques, including areas like people, process, technology leverage, organization, leadership.

- emphasis best practices based on successful solutions and lessons from real-life case studies.

- **be up to date and bring you detailed technical management knowledge reflecting current technology and business reality (such as the importance of scripting language, design pattern, and the reality of outsourcing).**
- **emphasize an approach that focuses on building long term success in projects - by improving process, people/team, infrastructures and core knowledge.**

Chapter 2

Vision

In [Yeh-4] one learns that Tao, the art of possibility, helps individual to find the meaning in life, and to answer questions like: Who am I? What do I stand for? What's my purpose in life? Where am I going? This art also helps organization, business, or project to find its vision, values and purpose.

In this Chapter, the centrality of vision in creating meaning and purpose in projects is emphasized and manager's role in helping the projects to focus on vision will be discussed. In addition, I shall also talk about the Capability Maturity Model (CMM), motivation, and project vision examples, as well as stories and practices related to vision.

2.1 Statements about Vision

Here are three statements. Do you agree or disagree with the statements, and what are your reasons?

1. **Vision is a corporate thing and is not applicable to small projects or individuals.**

2. **Corporate vision is our vision. No further translation is necessary.**

3. **Vision is an opportunity to create meaning in work for my project, people and myself.**

Here's my take to these statements -

1. **I disagree with the first statement. I think vision is also very important to local project and individuals.**

2. **I also disagree with the second statement. I think corporate vision needs to be interpreted in the context of local projects. This can help to make the corporate vision relevant and clear to project team members.**

3. **I agree with the third statement. I think vision is very important and useful in helping to create meaning in our work.**

2.2 Vision as A Base for Other Activities

Among the five arts, vision acts as the center, as vision gives meaning to what a person or a project is doing. Vision defines "what we are about" and "what it will be like when we succeed". Study of people who performs optimally (peak performance) [CSI] suggests that having a clear vision is the key to be purposeful and to be in "flow".

Perhaps because we are intelligent beings aware of our finite life, many are searching for meaning beyond oneself, through religion and work. Many great companies aspire to do good, such as Metronics' "restore people to full life". Those aspiration serves as very powerful vision and motivating force. Through vision and values one can find the work meaningful and one can be purposeful and fully engaged. Having a powerful vision and meaning have sustained people in most difficult circumstances.

The reason vision is so essential is because vision defines what the project and the organization is about and what it will be like when we succeed and reach our goals. Vision helps to create meaning and purpose for the project and vision helps team members to be purposeful in their activities. Vision also helps to provide the context for all project activities and give answers to questions like "why are we doing this?. Vision is closely related to values (what the organization stands for) and strategy (how to realize our vision?) and specific goals. But first of all, we must know where we like to be in the future, so we need to have a vision.

So to help create meaning and purpose for the project team members, managers need to talk about vision and treat it as a vital job. Vision provides the context for all the other project activities.

2.3 CMM Roadmap and Framework

In the software arena, many companies have adopted the Capability Maturity Model [CMM] from Software Engineering Institute as a roadmap for software process improvement. In CMM, a five-level progression toward process maturity has been proposed:

1. Level 1 - initial ad hoc process
2. Level 2 - repeatable process
3. Level 3 - defined process
4. Level 4 - managed process
5. Level 5 - optimized process

In the past, US Department of Defense (DOD) has run into a lot of difficulties about the quality of software delivered by suppliers. DOD has adopted the CMM framework as a way to assess the capability of software supplier organizations. DOD has made reaching a certain level of CMM maturity part of the qualifying criteria for suppliers to bid on DOD contracts. As a result, many defense contractors and many others have embraced reaching high level of CMM as the organizations vision and goal. So CMM certainly plays a big role in any discussion about software project management or improvement.

The five arts of business also provide a roadmap toward organization improvement. One can improve along any of the five dimensions of

the five arts iteratively and selectively according to the situation the specific project is in.

2.4 Using Vision to Motivate

One of the major role of project manager is that of a motivator. There are many ways in addition to a paycheck and corporate benefit packages that managers can use to motivate an individual on the project. In particular, project vision can be used as a powerful motivator:

- Help project members to feel excited about the vision/value of the project and hence help to create passion and meaning for work.
- Help project members to understand the importance of their work and how individual's tasks are related to the project and corporate vision.
- Help explain how individual's contribution benefits the whole project and how to align individuals aspiration with the project team's mission.
- Corporate vision usually aims to provide essential services to society or to bring out the best in innovation and technology. Help project members to see the benefits of their work to others.
- Adjust work assignment to include opportunities of development for the individuals, such as learning new skills, using new technology, etc. Support individual's career aspiration through project work.

- **Explore and help to realize possibilities for individuals through project vision and imaginative, flexible assignments.**

2.5 Some Project Vision Examples

Here are some examples of project vision:

- **Delight our customers.**
- **Help people having fun while doing great work.**
- **Innovation and technical excellence.**
- **Grow people and core competence.**
- **Be the best in our domain.**
- **Build and accumulate assets for the company - such as patents, tools, reusable components.**

Personally, I am a big fan of helping people to grow and to have fun while doing great work. It has been a guiding light for project teams I worked with.

2.6 Stories about Vision

2.6.1 Clarifying Vision for a Project

Frequently, at the beginning of a project, the needs of the customers are not that well understood. I had a small development project once,

just a few developers, for an internal customer. Initially, we understood the project as to display on intranet billing reports for international data services. The data set was quite large and could be organized in many different ways on screen for viewing. This particular customer seemed to be in a big hurry to get the reports even though many details about the layout of the user interface screens were not yet firmed up in the requirement. My developers were perplexed as they were used to work with customers who were very particular about screen layouts and graphical user-interface (GUI). A meeting was arranged for the developers to interact with the customer directly to understand what business problems the customer was trying to address. It turned out that the customer's main concern was to use the tool for revenue recovery. Many data service facilities for one reason or another were leased but not properly billed, so the focus of this billing tool was to help the customer's team to spot usage versus billing discrepancy quickly for various accounts. Nice GUI look and feel was not important as compared with getting the system up quickly to start revenue recovery. The customer also outlined many future extensions he had in mind so developers could plan the system accordingly. After this interview, the development work went very smoothly and quickly. The customer was very happy with the results and gave us more work later. The developers were also happy because they felt that the tool they developed were really useful and contributed to the company's bottom line. They were also happy because they were given opportunity to use

new technology like Java and web application - this project was done around 1997, when Java just came out and intranet web applications just began to become popular.

2.6.2 Don't Forget the Big Picture [CRA]

There was a project to develop a medical monitor accessory to display real time data in Operation Room and Intensive Care Unit. Key to success for the product is understood to have a compact unit and to meet cost of goods sold (COGS) target.

The project team worked according to project plan "by the book" and dealt with one problem after another, such as to get all firmware code in limited ROM space, to master display technology, and to work with third-party to collaborate on some pieces of specialized software. Project proceeded by tweaking here and there to overcome various difficulties.

While all members were busy to work on tasks, the project as a whole lose sight of the fact that the COGS of the product was creeping up higher and higher. When the product is finally almost done and the team was ready to celebrate, one member asked the question: "How about the COGS?" It turned out that the project had far exceeded that target and there was no way to bring it back down. It was a good product and was needed by the market, but in the end the project was cancelled because of the COGS cost was too high.

The lesson learned here is that there is a tendency to lose sight of the big picture in the heat of doing the project details. Some body needs to watch the big picture. Project managers need to keep the big picture visible, perhaps by using a score card, at all time.

2.7 Some Useful Practices in Vision

Help project team members to understand and support the corporate vision and values, understand the purpose of the project, and how the project supports the corporate vision and goals as well as customer needs. It is also important for project team members to understand how their work support the project goals, how their interests are in alignment with the project, and how their career goals are served by their assignments in the project.

I also found it helpful to meet with both the project team and individuals periodically, to maintain good communication and dialogue about the project vision, work meaning, individuals aspiration and any concerns or issues on people's mind. People may have concerns that their work is useless, boring, wasteful, or there are better ways to do things. Listen to them. It's real important that project manager helps team members to find meaning in their work.

Related to this, AT&T had an excellent practice which requires managers to meet with each direct report one-on-one once annually to review

corporate vision and values. This emphasis and effort helps people to understand and remember the corporate vision and values.

2.8 Sharing and Project Evaluation

2.8.1 Sharing about Vision

It is useful to reflect on projects you know and extract useful lessons or to share experiences with others. Here are the questions:

Most Useful - What have you found to be the most useful ways in creating meaning for your projects?

Toughest - What have been your toughest challenges in helping your project team to get excited about their work?

Here are some input from my own experiences -

Most Useful - As indicated in the story section above, it has been very useful for project members to have direct interaction with customers. Formal requirement is important. But in addition, it has been very helpful to have the developers listen to customers directly and understand how the programs they develop could help users and what business problems are being solved. In addition, many projects may start out as exploratory prototype tools. For these, interaction and

iteration with the customers is very crucial for the success of the prototypes.

Toughest - The most frustrating thing is to have project cancelled or redirected late in the project cycle. Frequently, people feel that their effort has been "wasted". Sometimes, some work may be salvaged and reused elsewhere. But mostly it's scrapped. Another challenge is to divide up the work so no one feels neglected. Most developers like to learn new technology and very few like to do maintenance work. The latter is however also essential even though not glamorous. I usually try to bundle maintenance work with tasks that would help the individual's career growth, such as learning newest technology, in the assignment.

2.8.2 Evaluation of Project Vision

Here are some questions one could ask about the vision of a project: Is it clear and well understood and supported by everyone? How about alignment of individuals with organization's vision? Do people find their work assignment meaningful and useful? What areas should be strengthened?

Another useful exercise is to contrast how projects may be different - the ones without vision from the ones with focused vision - how can one tell which is which?

Chapter 3

Timing and Logistics

Once the project team has a clear vision, the natural next step is to develop a strategy about how to realize the vision. There are unknowns in the future and many external factors are out of the project's control. So there will be many risks along the way that need to be addressed. In [Yeh-4], there are three levels of approaches in applying the art of timing to come up with a strategy - to predict and respond to the future, to predict and influence the future, or living in TAO and co-create the future. In this chapter, we shall discuss mainly how to predict and respond, such as logistics planning and risk management in projects. In addition, examples and useful practices will also be included.

3.1 Statements about Project Timing Issues and Logistics

Many people believe that software is much more complex and much less well defined as com-

pared with hardware. Then may be software is not amenable to the usual safety and logistics analysis for making hardware products. The following two statements try to focus our attention on whether this is true. Do you agree or disagree with the statements and why?

1. **Computer science is still a very young science as compared with, say, physics. Software engineering is being asked to tackle new and bigger problems with or without good theoretical foundation. Thus we may not know what is the correct margin of safety to use in these new applications. There may not be a "standard design" for many parts of the software system.**

2. **Software is easy to change and do change continuously, so software timing and logistics issues are fundamentally harder to manage than, say, military logistics or business supply chain management.**

Here is my input:

1. **Because software is more complex, it would require even more margin of safety to cover uncertainties related to its complexity. Margin of safety is definitely needed when program failures or exceeding capacity limits in software would cause significant losses in life or property. However, there is a significant difference between hardware and software components. One can safeguard hardware component failures by adding redundant paths and redundant components in the**

design. One can not do that for software because the failure modes for software is fundamentally different from that of hardware components.

2. Software is frequently an important part to support military logistics or business supply chain management, and the part can not be harder than the whole. So software issues cannot be harder than fighting a war or running a business. The logistics and timing challenges for the latter are much harder as there are a lot more unpredictable factors than those usually encountered in software production, hence harder to make plan for contingency.

3.2 Some Software Timing Issues and Logistics Challenges

There are many types of challenges in project management that are related to timing and logistics. The following are some examples.

Configuration management could be very complex. How to support multiple sub-projects in parallel development sharing the same code base? How to track source code changes with each specific problems? How to manage multiple changes to the same code module but deliver the changes at different time, not necessarily in the same sequence as the changes were made? How to track what versions of source code is used in the various production sites?

The need to support multiple versions of

source code brings to mind a related challenge of supporting multiple development environments. There is a need for an environment for development, one for integration testing, one for system testing, and one for inter-system testing. There is also the need to be able to re-generate identical environments with customer's, where there might be many versions out there in the field, so that one can reproduce and resolve problems encountered by the customers. One may also need to run multiple environments on the same machine in order to reduce cost. That would bring in other related problems such as changing or sharing databases for different environments.

When a new version of software goes out, there are many logistical steps involved to minimize customers down time in operation. There may be a need to migrate the database if there are schema changes in data. New tools may need to be configured in the field and customers may need training for new features. Cutover of software that spans multiple machines may need to be synchronized carefully so that software in all machines involved are compatible. In case of unforeseen troubles, there also needs to be a rollback procedure.

There are also many timing and logistics issues in staffing and in production environment. Various expertise are usually needed to work on multiple projects. The timing of deploying various experts for a project may need to be carefully planned. Long lead time items for the project needs to be identified and acted upon so the item would be there in time. Timing to introduce new technology or tools needs to be planned carefully so it will not impact production schedule. On

top of above, add features modification, personnel turnover, new tools or operation environment, last minute customer requests, all add risks to the project.

3.3 Anticipate and Prepare for Risks, Changes, and Crisis

A big part of project management is risk management, change management and crisis management. A most important task in logistics is to anticipate and manage risks. One needs to anticipate some level of changes and plan contingency for them. As an example, the configuration and production cutover tasks mentioned above would become more challenging if one also needs to accommodate personnel turnover, or new project direction, or last minute customer requests at the same time. Always try to avoid fighting fires on multiple fronts. There is a tendency for bad incidents to reinforce and multiply (snowballing effect). Just like doing many things right can have a synergistic, positive, effect on the project, having several things go wrong at the same time could have devastating consequence.

One may need to reserve some capacity to handle changes and for margin of safety. In the military tradition, one does not commit all resources to a battle initially, as many times having a reserve would make a big difference in the outcome. If something unforeseen happen, applying the reserve could help the project to recover. For software production, one kind of reserve is the use of

overtime. Although one would prefer to reduce overtime to a minimum. For high availability one needs to have backup standby machines in case one machine should be down. One also need fast support for production tools that one rely upon such as database management system, application servers, workflow manager, etc.

Many systems were built with great cost but do not meet customers' needs. One could avoid that by rapid and iterative prototyping so that delivered products are what customers really want and there is no surprises to either party. Identify high risk areas and use rapid prototyping to learn and help making the right design decisions is an important method to handle any new or uncertain situations. It's a form of rapid, low cost, learning.

Many software systems are unnecessarily complex. One can minimize risks by simplify, standardize and automate both the software processes and the software product.

Some of the risk scenarios could be anticipated and acted upon. One good example is hardware failures, such as hardware system crash or data memory disk failures. Thus for systems that need to provide 24 hours, seven days a week (24x7) coverage, one could design in disk mirroring and hot stand-by machines for uninterrupted operation. To protect critical data one needs to perform regular disk backup and have copy of data stored externally (in fire proof building) as well.

For even more strict requirement on availability, one could have disaster recovery site for critical systems.

Other common risks include personnel

turnover, vacation, project termination or downsizing. One could cross train people so there will always be several persons who could support any given task. Good, accurate system documentation is essential to help new people to get on board quickly. Provide training to new people. Assign mentors to new people. For major downsizing or project redirection upper management help will be needed to re-assign people to other projects. Try to address all contingency issues within your project but don't hesitate to ask for help when you have a need.

Another kind of contingency is to make sure that the system built is flexible, easy to maintain and scalable. In short, have a good system architecture and stick to it. It is no secret that a good systems will continue to grow in features and number of users. Thus one needs a scalable architecture to handle higher usage volume without performance degradation. The architecture needs to be extensible so new features could be added on easily, preferably with minimum impact to existing customers. Some technology can support "table-driven" programming. Systems built with such technology allow some features to be added simply by changing some configuration scripts or could even be configured by customers. In the architecture, one also needs to plan for adequate capacity to support high performance. There needs to be monitoring tools so that additional capacity could be added as the needs arise. We shall address many of these issues in the chapter on system architecture (Chapter 5).

3.4 Could Configuration Be Unified or Simplified?

Both configuration management and architecture might be simplified by using an existing architecture framework or application platform such as .NET or Java 2 platform. As will be indicated in Chapter 5 below, Java 2 platform supports many services, such as messaging, transaction, legacy system connectivity management. All of these services used to require separate servers or systems but can now be provided in a single platform. In addition, many Java 2 application servers, such as JBOSS, also support failover, clustering (for scalability), and monitoring functions. For projects on such platform, classes from multiple applications could be deployed together on the same application server. This further simplifies the requirement on system configuration to support multiple projects.

Object-oriented approach and platform independent languages like Java help to make the software system more loosely coupled and less monolithic. The late binding of class objects allow remote distribution of latest version at run time and simplifies the complexities of managing customers' version updates. One could also adopt procedures such as no source code change unless the code change will be included in next release (always "rolling forward") to reduce complexities in tracking change requests and source code changes. Another simplification is to decouple code base when two products from the same code base starts to deviate from each other sig-

nificantly.

Given the potential complexities of configuration management and software distribution, it behooves the project managers to use the most convenient and powerful tools for such tasks. Having good tools also make programmers more productive and hence reduce both cycle time and system cost.

3.5 Examples and Stories

3.5.1 Some Stories on Risk Management

Peter Ting is a legend in AT&T software development. He was recognized as AT&T Bell Lab Fellow due to his tremendous contribution in building many key software systems for AT&T operation. His favorite development technique is iterative rapid prototyping using best people, usually Ph.D. in Computer Science, small team, and state of the arts equipments. Because the team is very capable and interacts with the users continuously, the product is usually delivered very quickly and meets or exceeds customers' expectation.

I have encountered a serious downsizing one time in my career. I have six employees and a number of contractors that I need to place. Fortunately, my immediate supervisor had good contacts and helped to place all employees as a block to another project. This is very important to us as we want to make sure all employees are taken care of. Also fortunately at that time was

the fact that software market was still quite good and I was able to help the contractors to find jobs within AT&T very quickly as well. But it was a scenario that was hard to plan for.

I also worked on a large network management project for outside customer (US Government). It's a contractual requirement that we need to plan for disaster recovery. As part of our routine testing procedures prior to cutting over a new version of software, we conducted system failover tests and disk sparing tests. We also had simulators to routinely simulate large scale network problems to make sure that our network management systems are working properly when flooded with massive network alarms.

3.5.2 Case Study on Risk Aversion

Here is a project story that highlights a risk management strategy by not taking on projects that one is not pretty sure one can succeed [PRO].

The project is in the domain of Application Specific Integrated Circuits (ASIC). In this market, most projects can not be delivered on time or without mistakes. The new approach is a new business model based on on-time delivery with no mistakes by having repeatable, predictable process and method.

The engineering side came up with 22 items as checklist for sales to use on decisions to accept or reject new contracts. Decision on accepting new projects will be made jointly by engineering and sales. In addition, if changes on specification were made by customers, cus-

tomers need to sign-off on cost and schedule impact on those changes.

Bottom line - as a result of the new approach, the organization achieved over 90% on-time delivery without errors.

3.6 Useful Practices in Timing and Logistics

Plan project in iterative stages to minimize risks and losses [ADE] - commit increasing resources only as high risks areas have already been dealt with in early phases. In this way, functionalities increase rapidly and risks decrease quickly as more resources are committed.

Reduce complexity and rooms for error. Simplify, standardize and automate. Flexible architecture allows table-driven changes and easy to add new features. Use good tools to manage complex configuration, or to distribute updates.

Have margin of safety and plenty in reserve in performance and capacity. Have room for expansion and a clear path for scaling up when users and usage volume increase. Spread expertise around. Use multiple staggered teams so no one would be burnout. Ask for help and reinforcement when needed.

Watch out for downward spiral on setbacks, one bad thing could lead to more troubles. Have reserves to hold the line and recover.

Have good processes and documentation. Provide training to new people. Have backups in all areas. Use scenario analysis and plan for known

risks, such as disk crash, hardware failures, etc.

3.7 Sharing and Project Evaluation

3.7.1 Sharing in Timing Issues and Logistics

Here are two questions:

Most Helpful - What have been the most helpful techniques to your projects in the areas of timing issues and logistics?

Toughest - What are the toughest challenges for your projects in change, crisis or risk management?

Here is my input:

Most Useful - I rely a lot on good people and good tools such as configuration management tools. But I think the main thing for me is to plan and prepare for contingency. I asked myself what are the weak links in the project? What could go wrong and what could I do to minimize the risks? For example, I always try to build project library and to start extensive project documentation and training so impact of personnel turnover could be minimized.

Toughest - Without a doubt the large scale downsizing is the most difficult one to deal with. I ask for help for that situation.

3.7.2 Evaluate Risk/Logistics Management in Projects

Here are some questions to ask to evaluate a project in logistics and risk management areas: Are major risks identified? Is contingency plan in place to deal with them? Is rapid prototyping or iterative development process used? Are there areas that can be simplified or automated? What works and what needs improvement?

Also, as an exercise, contrast a project with little logistics support or risk management with projects that manage risks well. What are the differences between the two?

Chapter 4

Leveraging Software Manufacturing Infrastructure

After setting a vision (the art of possibility) and finding a strategy (the art of timing), the natural next step is to implement the strategy and realize the vision. This is where the art of leverage comes in.

As will be explained in this and next chapters, the project can be done more effectively by leveraging technology and other resources. In this chapter, we shall discuss the opportunities in leveraging open source software and organizational resources to assemble powerful manufacturing platform to produce software. We shall also point out the powerful leverage in reusing people's training and skills through standardization and simplification. As usual, we will include examples and useful techniques.

4.1 Leverage Comes in Many Forms

In [Yeh-4], it was pointed out that one could leverage just about anything for competitive advantages: internal or external, market, customers, technology, cost structure, competitors. Furthermore, masters of leverage frequently apply multiple leverages simultaneously to create irresistible force, to change the balance in a overwhelming and favorable way.

Productivity of business has improved tremendously by streamlining the business process and use robots or software to automate many tasks. Tasks can be done faster, cheaper and with less error. This idea of business process reengineering can also be applied to software production process as well.

Within software, there are opportunities for automation or reuse within the software product itself, in addition to automating the software production environment. The topic of how to use leverage within the software product itself will be addressed in the next chapter.

There are also great opportunities by leveraging through size or scale, such as the use of industrial standards, corporate guidelines and standards, as well as industrial or corporate standard components.

Additional opportunities for leveraging include: tools, reusable components, integrated development environment (IDE), open source solutions, people's experience, system or architecture frameworks, innovation by others (including com-

petitors), etc.

As pointed out in [Yeh-4], leverage is most powerful when applied in multiple areas simultaneously. The synergistic "market inflection point" effect of applying multiple mutually reinforcing levers can help to elevate the project to operate at a much higher level. Each lever applied helps to reinforce and to stabilize other levers and prevents any lever from sliding back.

4.2 Statements About Leveraging Software Manufacturing Infrastructure

Here are two statements. Do you agree or disagree with them and why?

1. One can not use open source freeware due to its lack of support.

2. Each project is unique and different. It is unrealistic to use the same development environment across projects except for very closely related projects.

Here is my input -

1. For popular open source software, one can usually find commercial support for a small fee. Many such software are actually quite reliable due to heavy usage from software community. In contrast, proprietary software may not have been exercised as thoroughly due to much limited usage or testing.

2. Many tools of the development environment are quite general. For example, tools for object modeling and analysis or language related packages such as compiler or debugger are all very general and can be used in many different projects. Each project may need other tools specifically for their projects in addition to the generic development environment framework.

4.3 Build/Assemble Your Own Software Factory

Automation has been a major driving force and a powerful leverage in improving business productivity. Many companies continue to deploy information technology to improve and automate their business processes. This approach is equally applicable to the processes of manufacturing software. Some would call such a production environment a software factory. There are many areas one can automate the software production processes through tools. The following identify some of the areas where tools are available:

- Project planning, management, and quality control tools.
- Problem domain-specific tools (such as artificial intelligence, imaging, or signal processing).
- System analysis, object modeling tools.

- Integrated development environment - software design, coding, unit testing, debugging, logging and tracing tools.
- System build, source version control, problem reporting and tracking tools.
- Integration testing, system testing, interface/protocol simulation tools.
- Performance monitoring and analysis tools.
- Software distribution, deployment tools. Remote monitoring and maintenance tools.

4.4 Leverage Open Source Software

4.4.1 The Advantage of Open Source

Try to avoid reinventing all the solutions or redoing all the work in each project as that would cost too much and take too long. Try instead to leverage from other's results, especially since solid tools are frequently available for free on the internet[OSI]. Many toolkits and libraries are provided for free, such as Java SDK from Sun Microsystems. Sun also leverages software community to solicit input to new features for many of its products. Microsoft gives software developers early beta version for new products so software community serves as free beta testers.

Since many software pieces are needed for a comprehensive software development environment, it used to be very expensive to put to-

gether one such software development environment. There are also concerns about compatibility between tools from different vendors, especially if any of the tools use proprietary protocols or interfaces. Recent development in common standard and open source software [OPE], especially for hardware platform independent language such as Java, provides significant leverage in both reducing cost and addressing many of the compatibility issues. By using open standards one could swap out a tool made by one company, such as an application server, and replace it by similar tool from another, without losing capability or compatibility.

In the past, people stayed away from freeware due to concerns about support and questions about the quality of the software, especially concerns about software virus. The recent trend is that popular freeware are much more heavily used through massive free downloads and hence the bugs are shaken out more thoroughly than many of the proprietary software products, provided that one avoids the very early alpha or beta versions, when the product is still being debugged. A good example is the LINUX operating system or the Java software development kit from Sun Microsystems. Both are very robust. Some popular tools supporting development and application in Java 2 platform include ANT (build tool), STRUTS (Model View Control framework for web application), JBOSS (application server). Many other tools can be found in [OPE]. Another concern is maintenance support. If one encountered a bug or urgently needed some new features, where could one get help? Even though one has

access to source code for the open source software, it is not attractive to need to go in and change the source code and to maintain it. For many popular software, one could usually find commercial support, training and consulting for a small fee. If one can get robust tools for free, why buy?

One can get a feel for what it's like to do an open source project by participating at Tigris [TIG] web site. It hosts and supports many open source projects by providing tools and environment - such as source code control tool, issues tracking, mailing list, discussion forum - to make it easy for anyone to start a new open source project.

Sometimes, many hundreds can participate to an open source project. The complexity that comes from large number of participants and contributors in the "team" is minimized by having a core team and "owners" who control/coordinate features, with each person for specific areas. The process and tools environment for open source projects tend to be relatively simple. Since most are volunteers, target dates for required new features are harder to pin down.

Personally, I think the open source movement is a great boon to the software community and public at large. It's a great spirit to build up community assets that increase everyone's capability. Everyone is enriched and enabled in the end. It's a kind of free wealth building. Once the tool is there, every one can use it, for free, and the wealth is not diminished, but multiplied. This is because tools are built by using other tools. The bigger the base, the easier it is to add more tools to the heap.

4.4.2 ANT - Another Neat Tool [HAT]

If an organization is new to open source software, perhaps one can ease into using it by using some tools for the software production environment first, instead of jumping right in and using the open source code inside the software product itself.

As an example of these free tools, let me talk about one of them, ANT, a build tool. Able to build system automatically and quickly is crucial to the success of a project, especially when the project gets big.

The free tool ANT (Another Neat Tool [HAT]) is aimed at supporting build in the Java environment. It has a simple syntax, with three-tier hierarchy (project, target, task). People with background from UNIX may be familiar with Make or Nmake build tool. Unlike Make, which is file-centric, ANT is task-centric. Compile (javac), package (jar) are both built-in tasks. Because new tasks could be added (ANT is extensible), ANT is more flexible and can handle things like deployment, documentation generation, work-flow, website maintenance, quite easily. It can basically be viewed as a task engine. Some other custom ANT tasks include filter, logger, mapper. Make script needs be modified if new files are added. In ANT, adding new files by itself would not need to change build script as compile or package would do all the files in the directory. ANT runs very fast and also has built-in support for Java 2 platform, such as enterprise java bean (EJB) compilation and packaging.

ANT works well with many popular software

configuration management (SCM) systems such as CVS, ClearCase, SourceSafe, etc. ANT supports unit testing in Java easily with JUnit and could simplify and automate software product deployment. For Java project, ANT complements front-end integrated development environment (IDE) well and has good support for back-end team project tasks such as building, testing and deployment. Some ANT-aware IDEs include jEdit, IntelliJ IDEA, SUN NetBeans/Forte, IBM Eclipse. IDE plus ANT and SCM would provide a basic framework for software manufacturing environment.

4.5 Leverage Organizational Resources

It's a good idea to comply with industrial standards for both product compatibility as well as to avoid being locked-in by a particular vendor to their proprietary products.

Due to the importance of information technology, there is usually organization-wide standards and guidelines, usually under Chief Information Officer, in areas like development environment, tools and components. Following such guidelines would help to reduce cost, simplify training and enhance reuse.

Just like Southwest Airline [YEH-4] gained tremendous competitive advantage by using only one kind of aircraft, cost of software production will be greatly reduced if one can standardize the platform, language, tools, components

used across corporate product lines. With standardization, people could be moved to different projects with little need for retraining, as the production environment and software programming skills required in different projects will be identical. Along the same line, the whole industry gains tremendous saving by agreeing on standards instead of having many incompatible products, protocols or conventions.

In development environment, it is preferable to buy instead of build, as software environment tools are not the focus of your software expertise or product line. For example, why write your own converter to map between java objects and XML representation, when free commercial packages are available?

For your own specific problem domain, there may be opportunities to accumulate and share various domain-specific assets that you built - tools, code libraries, objects, and other reusable components.

4.6 Stories in Leveraging Software Environment and Components

AT&T invested significant resources to evaluate various outside software packages and components for various types of applications. (For example, whether the application needs to support many users and heavy traffic volume or just a few users and light traffic, and whether the system must be available and robust all the

time (24x7) or just needs to be up during weekday working hours.) There is a comprehensive Foundation Architecture that was supported by many experts across organization and updated periodically. It provides guidelines of what type of components or packages (both hardware and software) could be used for various type of applications. It is a wealth of information and a great resource for individual projects in the company. Exceptions are allowed but must be requested and project needs to explain why exception is needed. With the Foundation Architecture the task of hardware/software selection for projects is greatly simplified. AT&T also benefited greatly by reducing the great variety of software or hardware products that were deployed before the standard is in place.

For the projects I worked on, many free Java tools were used, such as ANT, Java SDK, STRUTS. However, we still stuck to commercial products like Relational Database Management System (RDBMS), workflow engine and application server.

As time goes on, the pressure of software license fees become even more of a burden as more tools are needed for the increasingly more complex software. In some projects, software license cost is already bigger than hardware cost. In time, I think more and more projects will be using popular open source freeware, especially when some of them are doing aggressive marketing, such as JBOSS, which offered free porting from the popular but expensive BEA's WebLogic to JBOSS.

4.7 Useful Practices in Leveraging Software Environment and Components

Here are some suggestions:

- **Follow and support corporate guidelines about development environment and asset component reuse. If no such guideline exists, consider to start one for your organization.**
- **Automate as much as one can as any task automated will be done at very low cost and with no human error.**
- **Research the world-wide web (WWW) for robust free tools or solutions that could meet your needs. Try them out and consider using them to reduce the cost of your development environment. WWW is truly an amazing place where hundreds of millions of intellectual flowers are blooming. With powerful search engines, one can quickly find the few nuggets that one needs. Of course, not everything one encounters in the internet is trustworthy or useful. Careful evaluation is needed. But it is still a great saver of time and effort in most cases.**
- **Don't build if you could buy or borrow (reuse) internally.**
- **Watch for any asset you built that might be reusable. Share them with other projects in your organization.**

- **Use and follow industrial standards.**
- **Simplify and Standardize - could one use less variety in everything?**

4.8 Sharing and Project Evaluation

4.8.1 Sharing on Software Environment

Two questions:

Most Useful - **What are most useful to you in setting up your software manufacturing environment?**

Toughest - **What are the greatest challenges in the area of software manufacturing environment for your projects?**

Here is my input -

Most Useful - **It has been most helpful to have AT&T's Foundation Architecture to guide my projects in both hardware/software and architecture/components selection as well as in tools for development environment.**

Toughest - **Sharing environment for several related projects on the same hardware has been a big challenge. We were limited in hardware so sharing is required, but configuration for various applications is not always compatible. The more recent Java 2 platform would have eased this problem about sharing.**

4.8.2 Evaluation of Software Environment

Some questions to ask when evaluating software manufacturing environment - What are its strength? What areas need to improve? Is your "software factory" fairly complete in using tools for automation? Are there opportunities to use open source tools? Are we following corporate guideline in tools and environments? Are we applying all the levers to gain the synergistic market inflection point effect?

A useful exercise is to contrast projects that use no leverage in manufacturing environment with those that maximize leverage in this area. How to tell the differences between the two?

Chapter 5

Leveraging System Architecture Framework

In this chapter, I will emphasize the importance to leverage experienced architects to conduct system architecture reviews, as well as productivity gain through leveraging component-based system design (such as in Java 2 platform), design patterns, pattern-based architecture framework, and advantages in using domain specific scripting languages. In addition, provocative statements, examples and useful techniques will also be included.

5.1 Statements Related to Leveraging Architecture Framework

Here are two statements to think about. Do you agree or disagree and why?

1. **Graphical User Interface (GUI) could be built easily with many reusable components (button, list, scroll bar, etc.) but this won't work for other areas which are not structured as a composite of components. The business to customer (BtoC) web applications could be built with design pattern such as MVC (model-view-controller), but it is not clear that the pattern-based architecture framework could work elsewhere.**

2. **One constant thing about software is that if it is successful, there will be more features added to it, so one better designs ease of maintenance into the software right from the beginning.**

Here is my input about these statements -

1. **In fact, both component-based approach to software construction and pattern-based architecture framework have been applied to back-end (workflow, connector architecture) as well as to middleware [SCH]. So the application of these approaches turn out to be very general.**

2. **I think to make software easy to maintain and easy to add new features is a very important architecture consideration and a key**

factor to reduce software cost over the long term.

5.2 Some Architecture Concerns and Design Principles [FOW, VAS]

5.2.1 Key Architecture Issues

In designing the architecture of a software systems, there are many concerns in addition to meeting the specific feature and functional requirement. Here are some of the key architecture issues:

Reliability and availability - Will the system perform reliably and always give correct answers? Will the system be able to withstand various failures - disk failure, power failure, circuit board failure, site disaster - with no corruption on data and no downtime in operation? Will the system be available without excessive down time during maintenance?

Performance and scalability - Can the system meet the expected throughput and response time? Can the system meet increased volume or number of users without the need to change the system architecture or suffer performance degradation? What is the expected increase in usage in the future and what is the scenario for system upgrade?

Maintainability and flexibility - How easy or difficult it is to perform the operation, administration and maintenance functions for the system? How easy or difficult it is to add or modify system functionalities?

Security - How secure is the system from hackers, computer viruses and worms? What kind of protection are there for sensitive data and sensitive functions in the system from unauthorized use? Has the system been designed by following organizational standards and guidelines on computer and network security?

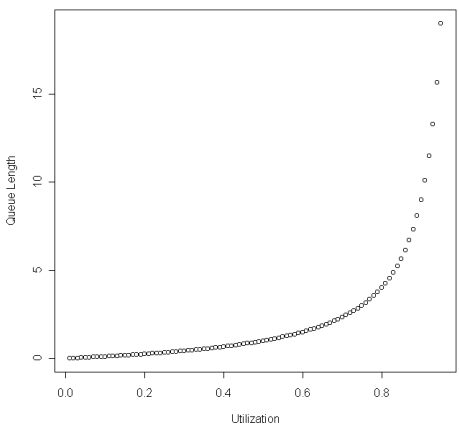
Internationalization - Does the system need to support global customers as well as interfaces in multiple languages? If so, has such internationalization capability already built-in within the platform and architecture, such as using Java's Unicode support and internationalization package?

Ideas to address many of the above issues will be discussed in the next few sub-sections.

5.2.2 Enhancing Performance and Capacity

Performance is closely related to capacity. A simple lesson from queuing theory [ROB] to keep in mind is that for service requests that arrive randomly one can't utilize the capacity to the full extent without causing delays or performance degradation. Let u (utilization) be defined as the ratio of load (service request rate) to capacity

(number of requests the system can handle in one unit time). If requests should arrive only at regular interval, one would expect that there would be no queue or waiting period as long as the load (arrival rate) is less than the capacity. However, arrival tends to be random and sometimes could occur in burst. For simple queues, it could be shown that the queue length (or wait time) is proportional to $\frac{u}{1-u}$. This means that the wait time or response time could get very large when u approaches 1, or when the load approaches capacity, as now the denominator in the formula approaches 0. Queue length is plotted against utilization (u) in the figure here. As illustrated in the figure, queue length gets large as utilization approaches 1. This phenomena could apply to many types of capacity, whether it is communication bandwidth, CPU processor time, or other exclusive resources such as locking data in objects or in disk for exclusive write, as long as the arrival for service is random.



The requirement for performance and for reliability frequently work against each other. To get great performance one would like to be able to do many tasks in parallel, but to get great reliability, one may need to do things in sequence, and sometimes lock up data resources for exclusive use. To protect data integrity, there may be a side effect on performance. The more extensive the locked operation, the more severe is the performance impact, so more activities need to wait. Locking a whole table in database will have more severe impact on performance than locking just a row of data in the table. So there may be a tradeoff between performance versus reliability. One example is whether dirty read of database is allowed. Dirty read allows another application to read the data while the data is being updated. It's fast (no waiting) but the data read may not be internally consistent. In some of the workflow scenarios for ordering systems, multiple set of data may need to be "rolled back" when an order is cancelled or modified. For complex data structure this could become a significant challenge in architecture analysis to support such changes that would maintain both the data integrity and to minimize performance impact at the same time.

Another useful concept in performance is to find the bottleneck or weak link in performance. We don't want to use a weak ring in a chain link because the strength of the link is determined by its weakest member. We don't want to use a thin pipe to connect between two bigger pipes as the flow rate of the pipes is determined by the narrowest one. In electronics, there is a principle of impedance matching, so the flow of elec-

trical energy (power) between circuits could be maximized. In computer systems, disk controller tends to be the bottleneck in data access [COC] as it is the only place where slow mechanical motion is involved. The advantage of fast data transfer speed in computer boards or network will be lost if access speed to data in memory disks is too slow. There are various techniques to reduce the impact of this limit, such as to distribute data over multiple disk controllers for parallel input/output access and using buffers in core memory for frequently accessed data.

In the past, hierarchical structures (such as subroutine libraries) were introduced to control complexity. Object oriented approach combines procedure with data to isolate object interface from object implementation. This also provides good support to data integrity control as access to data is restricted through the interface. However, for performance and several other reasons, popular database management systems (DBMS) is not yet object-oriented. DBMS are still dominated by relational database. Relational database have been proven to be highly reliable for very large and distributed databases. Query for various reports are easy to formulate without the need to worry about how to navigate the access path of the physical tables. So there will continue to be a need to map between objects used in application that reside in core memory and the persistent data storage in the relational table of database. Many tools are available to automate the translation between the two. For performance reason, there may also be a need for multiple copies of the same data, loaded in as different objects, to

reside in the core memory. So when there is an update to the database on this data, there is a need to keep all the copies of object in sync with the update.

As mentioned before, there is also a tradeoff between performance and accuracy, by deciding whether dirty read during write is allowed [FOW]. As for the database itself, one needs to distinguish between a historical snapshot that is internally consistent (e.g. end of month archive), from the on-going dynamic state at the moment [VAS]. Different needs can best be served by using different versions of the database. Market analysis may be best performed by yet another kind of database, the data warehouse, where many special attributes can be collected just for the purpose of in-depth market analysis.

When usages (traffic, users, orders, transactions) have increased, system can be scaled up easily provided that data from one user or one order is pretty much decoupled from another. May be the user or order specific data could be keyed off by the user name or order number and put into a single table. Database schema should be designed to maximize such separation so performance would not be degraded due to heavier usage. Namely, it is preferable that the locking of one set of user or order data would not affect the read/write of other user/order data, hence create no bottleneck in data access. However, when the data or objects from individual users/orders need to interact in some ways, the complexity of the architecture goes up very quickly with large user/order volume. This is the case for ordering and provisioning for many telecommunication ser-

vices.

5.2.3 Enhancing Reliability, Availability and Flexibility

In hardware engineering, redundancy is an important principle used to enhance reliability, because it is very unlikely that multiple components would fail at the same time. Redundant components and redundant paths are used to improve system robustness so no single failure would bring down the system. Preventive maintenance is also useful for hardware. Hardware devices have a life expectancy in usage and could be replaced before the expected life time. These ideas, however, have not been that useful in software. One could compute a problem in two different ways and compare their results. However, this makes the software more complex as one more way is not only costly but also introduces new possibilities of failures.

In hardware reliability one also uses failure mode analysis, namely, in what ways the system could fail. This is also a very useful exercise for system analysis including both hardware and software. Through such analysis, one could identify scenarios for potential problems and modify the system architecture to minimize system down time. One such scenario is whether there will be system down time if a circuit board failed? In some systems, there is no way to take out the bad part without shutting down the system, so that would become a system down time. Another example of such failure mode analysis would be the scenario of a second failure before the pri-

mary failure has been fixed, say, either in mirrored disks or in the hot standby machine. Is it important for the system to be able to withstand such consecutive failures? Consequence of power failure would be another important failure mode. Are there emergency power available? Here, system availability is related to system reliability. System availability is enhanced by making the system more reliable. In addition, availability will be better if system could be repaired or routine maintenance performed (such as data backup) without the need to shut down operation.

Some computer hardware now supports hot-swapping of circuit boards, so one can swap out and replace circuit boards without stopping operation. A somewhat related situation is the ability to update software modules in production. In many web-based applications system upgrade become very simple as the web client would simply access a server to retrieve the latest version of objects. In some software applications multiple version of the software could run in parallel, so that a workflow that was started with older version will complete with the older version, but new orders or cases will be processed by the newer version of the software.

Software industry has made great stride in supporting system flexibility. Java language supports late binding, dynamic loading and linking of objects and run time determination of the properties of objects (by using Java bean introspection properties). In addition to dynamic invocation of objects, the CORBA standard also supports generic data type such as the type ANY [ORF].

More help on system flexibility is provided by

design patterns and Java 2 technology platform. We shall see an example in the following on how indirection and translation helps to make the system more "table- driven". We'll also see that pattern-based architecture framework allows parallel, incremental development and deployment of feature package "slice" and hence supports a more flexible development process.

5.2.4 More Ideas and Solutions on Architecture Issues

System Availability - This is helped by disk-mirroring, failover support (hot standby), hot-swapping, backup power supply, system management functions to monitor hardware, network, and application integrity.

Scalability - Need to have clear expansion path for more core memory, disk memory, CPU boards, I/O connection, more hardware in systems. Use tools and techniques of clustering and load balancing for capacity expansion.

Data Reliability - Use locking for exclusive write access.

Data Mapping - Need objects for separation of concerns. Need relational tables for efficiency and ease of data retrieval. Bridge object-relational transformation either by custom code (for example, one simple way is to have all data of objects in an inheritance hierarchy contained in the same database table with flags to indicate which fields

are used) or use commercial or free conversion packages.

Use Multiple Data Images - Use one snapshot used for the end of month report and a separate image for real-time updates [VAS]. Use yet another image with additional data fields for data mining or marketing analysis (data warehouse).

Keep Data Separate - Data from different orders, customers, transaction might be separable to a large extent, so locking on one order or one customer would not interfere with locking on other orders. (Row locking.)

Report versus Data Processing - Objects needed for data processing (for an order) are very different from data needed for reports (objects from all orders that meet certain conditions). Allow dirty read or allow some stale data to be used could greatly speed up things. Otherwise, all order processing (writes) need to halt while read tables for reports are in progress.

Concurrent Access versus Data Safety - Keep data in core (caching), keep multiple copy of objects, use multiple threads (concurrency) can improve performance. However, need to keep data copies in sync with updates and avoid multiple writes on same data (synchronization, locking) and avoid deadlock.

Business Transaction with Rollback - Business transactions may span multiple machines

and databases, and also need to be ACID (Atomic action, Consistent state, Isolated from view, Durable and survive crash) [FOW]. Two-phase commit protocol and rollback capability are needed to support this data integrity.

System Flexibility - Use late binding of objects, dynamic loading, name indirection, flexible data structure, and table-driven techniques for ease of addition and modification of objects and features.

5.3 Architecture Patterns and Styles

There are many different type of architectures [BUS], I'll mentioned just a few here:

- Sequential - Most people learn programming design this way: first do A, then B, then C, all sequentially. UNIX Shell has pipes/filters that connect small commands in a sequence.
- Layers - An architecture related to pipes/filters is the layer architecture such as typically used in implementing communication protocols where messages are decomposed and processed by consecutive layers to handle different aspects of the communication tasks.
- Workflow - In many applications that support business process automation, tasks flow through the system according to various work flow. It's more complicated than a single sequential flow pattern as various flow

paths could meet or disperse from nodes in the flow chart according to the specifics of the task.

- Distributed System - In object or message-centered systems, one needs some central agent - such as object or message broker - to distribute information between distributed objects or message queues.

- Event Driven - In Graphical User Interface (GUI), typically, system is user-driven. System is typically consisted of a big event loop. If some buttons or keys are pressed, a corresponding event is triggered, and system will catch those events and respond to them. Otherwise, system sleeps and waits for the next period to see if there is any pending event. Client-server applications are similar. Servers are waiting for clients to call and request services. If there's no callers, servers will sleep and wake up in the next cycle.

- Blackboard - many Artificial Intelligence (AI) or expert systems are consisted of three pieces, the engine, the expert rules and the data. The engine will check if any pattern of the rules are matched by the data. If so, the rules are "fired" and action of the rules are carried out which may update the data, and which in turn can trigger further rules firing, etc.

- Web-based client-server applications - client requests come in and are processed by a controller. Requests were sent to

Model/objects for processing. Results are used to generate the next View and sent back to clients. One such architecture pattern is the Model-View-Controller (MVC), where more details will be given below.

Architecture concerns may be somewhat different for different type of architectures and applications. For sequential architecture, concurrency may not be an issue. But for web-based application, supporting many concurrent clients, data locking and concurrency control may be very important.

5.4 Leverage Architecture Review

From the discussion above, architecture design could be quite complicated and there are many issues to address. Since typically not that many people in the project have done many projects or architecture designs, it is useful to leverage the architecture expertise from many projects. One way to do that is through an architecture design review. This has been an important approach in AT&T software community. Experienced software architects are identified and considered a corporate-wide resources. They participate in architecture design reviews for many projects. To improve the quality and effectiveness of these reviews, some additional actions were taken that have been proven to be very useful at AT&T:

- Provide Training in Architecture Review - A course is developed to instruct people

about what's important in architecture reviews, the typical architecture concerns and solutions, and how to conduct architecture reviews.

- Guidelines - An Architecture Review Handbook is developed, where for each architecture concern area, a comprehensive list of questions are identified (see below).

- Preparation for Review - A most important step before the review is for the architecture design team of the project to prepare and work through the list of questions. As a result, many details such as capacity margin, expected performance and response time, failure modes, alternate architecture designs considered and their pros and cons, etc. will be clarified.

The architecture training and architecture review focused on three areas: error recovery, operation, administration, and maintenance (OA&M), and performance. These three areas seem to be the place that differentiates a prototype from a robust system. Here's a few sample questions to give a flavor what such a checklist might be like and the projects need to prepare and answer these questions at the review:

- How quickly can the files or database be backed up? Can this be done while the application is running?

- Does the system include monitoring mechanisms that would send alarms when criti-

cal errors or performance thresholds are exceeded?

- What is the resource budget for each function/process in the system?
- What are the response time and throughput requirements, and what are the load requirement both for normal load and peak load: number of expected users, network bandwidth, and system traffic?
- For testing, can one save a snapshot of the system and then resume from there? What needs to be saved?
- Can more than one version of the application run on a single machine? Can other applications share the same hardware and database?
- What are the failure modes of the system? How are they handled when failures happen?
- If the system is required to support feature loading or customization, what techniques is used to support this? How to ensure the correct set of features are turned on for each customer?

For a list of design review questions focusing on applications in database environment, see [INM].

5.5 Software Engineering with Component Rich Java 2 Platform

5.5.1 Leverage Technology - Reusable Components, Design Patterns and Architecture Framework

In this and next two sections, I shall focus on the importance of reuse by leveraging technology to reduce the work needed in software system and architecture. One example is reusable component-based class objects as exemplified in using the Java 2 platform in solving web-based client server application problems. (Java 2 platform includes Java 2 Standard Edition [J2SE], Java 2 Enterprise Edition [J2EE] as well as a micro edition for hand-held and other smart devices.) Another example in technology leverage is to collect related class objects into design patterns and use them as larger building blocks. And finally, one could also assemble design patterns into generic system architecture framework for reuse at a grand scale.

Reusable components has been a dream of object-oriented approach in program construction for some time and today it has finally become a reality. Design patterns complement algorithms by focusing on solving recurring structure problems [BUS]. In addition, by linking these design patterns one can frequently arrive at an architecture framework for various problem domains [ALU, ADA].

We shall get into more detail in these approaches below. If you need to come up with architecture for your problem domain, it is certainly worth checking to see if you could use existing components in J2SE or services in J2EE and assemble appropriate design patterns to come up with a viable architecture framework for your domain.

5.5.2 Java 2 Platform

Java 2 platform standard edition (J2SE) is feature rich. It has over 3000 classes and supports lots of features, including GUI, event handling, exception handling, input/output, multithreading, generic and powerful data structure such as collection, networking, DBMS connectivity, remote object invocation, security, XML, internationalization, CORBA support (Java IDL), etc. In addition, Java 2 is also hardware platform independent [VEN], robust, supports dynamic loading and linking of class libraries.

Java 2 Enterprise Edition (J2EE) [J2E] took the approach to solve web-based client-server application problems by providing lots of building blocks and make it easy to assemble them. Many important services are more difficult to program. J2EE provides them so programmers don't need to worry about them. Services provided in J2EE include HTTP protocols, transaction, messaging, security, interface to database, and remote transport of objects. Lots of components are ready to be used. J2EE also eases software distribution, deployment and upgrade. Sun Microsystems developed Java 2 platform as open standards

through community-based process so compatibility across vendors' products would not become a problem for users.

Object-oriented languages are around for a long time. But component-based software construction was not practical for quite some time. There were few reusable components and people are concerned about using them for various reasons. This situation started to change in GUI, where many components were built and can be assembled easily to build GUI through tools like Visual Basic and X-Window toolkits. With Java, especially the Java 2 platform, the reusable components picture has changed dramatically, as every type of application or services is now supported by lots of components and available for free through open standards like J2SE or J2EE platform. As I shall discuss more below, with J2SE and J2EE, we now see real reuse become a reality at multiple levels - components, design patterns, and pattern- based architecture framework.

One consequence of this massive level of reuse is that programmers' productivity is greatly enhanced. With objects, one person can start to manage larger chunk of code as classes which hide a lot of details and enables reuse. With reusable architecture framework such as MVC and standard solutions on scalability that comes with application servers, one can put a web service up very quickly. When one thinks about it, a web service is quite a complicated application. Not only is each screen fairly complex with lots of widgets in it, one also has to generate dynamically new web pages based on user data and ac-

tion. And furthermore, one may have to handle high volume of traffic and heavy load, as there may be hundreds or even thousands of hits in a minute for a popular web site. All this become doable by one person or a few people, and can be done quickly, when many solutions, services, and components are available for reuse.

5.6 The Rise of Design Patterns in Software Engineering

In the past, computer science and software engineering have focused more on algorithms, data structures, and specific technical issues such as design of graphical human-computer interfaces. This is changed with design patterns.

Structure patterns for building architecture were first identified as an area of study by Christopher Alexander [ALE], and subsequently brought over to software, and made well known through the book of "Design Pattern" [GAM] and others. Design patterns focused on solving structure patterns needed in a general software system and hence complement the past focus on algorithm. Design patterns support reuse and the assembly of larger structure such as architecture framework. One can view software systems as building up from class objects into design patterns, and from design patterns into system architecture. And this architecture is used to embed the algorithmic solution of the particular problem the system addressed.

There are many design patterns to choose

from. They serve to provide standard solutions to many "structural" problems frequently encountered in software systems. To give a flavor of design patterns, here are some design pattern examples [J2E, GAM, STE]:

Composite View - Many web pages is composed of several sub-views such as banner, footer, control panel, etc. The composite view pattern supports the construction of a composite view from basic views by making composite view both a derived class and an aggregation of basic views.

Facade - Facade is a very useful pattern when one needs to hide the individual interfaces to a group of complex components so that the client could only access the group through the interface provided by the facade.

Singleton - This pattern provides a single instance for general access to a class. In J2EE, the service locator is implemented as a singleton. Any client needs to locate services from remote objects just need to invoke the service locator to find them.

Adapter - This pattern enables two classes with incompatible interfaces to comminicate through the class adapter. When the client makes a request, it gets translated by the adapter into commands that the adaptee can understand and execute.

Proxy - This pattern can work as a surrogate object or gate keeper to control the access to the real object. It is frequently used to implement access security to make sure that the clients have the right credentials and permission before they are allowed to talk to the real object.

Iterator - This pattern provides a consistent way to sequentially access items in a collection. Client works with aggregate (collection) interface which can return a concrete iterator for client to use. The concrete iterator is derived from the abstract iterator interface that can manipulate the collection aggregate, such as first, next operations.

Factory Method - The factory pattern supports the creation of multiple concrete products from one abstract product interface. The particular products to be created is determined by instantiating different subclasses.

Abstract Factory - This pattern is similar to the factory method pattern except that the abstract factory pattern allows the creation of multiple family of related products.

Command Pattern - This pattern separates the invocation of the command (command request) from the execution of the command (command action). Java event handling model supports such separation so events triggering could be separated from event handling or processing.

Command Factory Pattern (Table-driven Programming) - Let us now combine the command pattern and the factory method pattern into a command factory pattern to illustrate the how to do flexible, table-driven programming by using these design patterns. See chapter 2 of [BER] for example and code.

Let us assume that a hotel reservation system support "Add" and "Delete" request buttons on a web page. These commands can all be derived from a generic command interface (command pattern). In the server, each different command key word could be translated from a script file (table driven) into a different class object, which could be generated dynamically (factory method pattern). All commands support an "execute" method when invoked, but the resulting action will depend on the command. With this approach it would be very simple to add a new command such as "Change". One just needs to add a button to the command menu, and add a line in the translation script, and having the Change command class compiled. One does not need to recompile the system or change the "Add" or "Delete" classes in any way.

5.7 Assemble Design Patterns into Architecture Framework

There are many examples in books [ALU, BER] and web sites [J2E] that show one how to assemble design patterns into an architectural framework. SUN's pet store example in the

blueprints subdirectory is particularly helpful. The core J2EE patterns catalog is a good place to look at how various design patterns already provide a fairly complete architecture framework for web-based client-server application.

Let me use the MVC pattern to illustrate the concept. The web front-end application typically has three functions, model, view, and controller (MVC) as indicated in MVC pattern or fleshed out into the architecture framework STRUTS. The overall flow is like this. The controller, after some filtering such as making sure user has logged in, makes decision about where the client request should be sent to be processed by various system objects (modeling). After the modeling part is done, there are two tasks remains, namely, what screen should be generated and how to incorporate the data coming back from the modeling to generate the view. A number of other patterns, some mentioned earlier, can be used to perform these tasks: dispatcher view, view helper, and composite view patterns.

There are many design patterns to ease the work for the back-end as well. There is a business delegate pattern, which makes local interfaces available to the clients for remote objects. The service locator pattern can locate remote objects. The session facade patterns can hide complex details of the database entities. There are also a number of patterns to support efficient database access and object database table mapping, such as transfer object pattern, value list handler pattern, composite entities and data access object patterns.

By assembling all these different patterns to-

gether, it is not hard to imagine that one can put together a web application architecture framework, including both front-end web access and back-end database interaction, fairly easily. Even though we only talked about the web-application front-end and back-end here, many other pattern-based architecture frameworks are also available [ADA].

To take advantage of all these reusable components and design patterns, it's very useful to look into if one could come up with such a reusable architecture framework for one's own application domain.

5.8 Could One Work at a Higher Level? - Problem Domain Scripting

J2EE and .NET platforms are very powerful. But at the same time, they are also very complex. Is there a easier way to do the same task? For special domain, one may be able to build a scripting language with powerful support so one can write solutions to problems naturally and easily. Two good examples are using awk [AHO] for text pattern processing and using S [BEC] for statistical data analysis.

Tcl/Tk [OUS] is a widely used, general, and extensible scripting language. One of its extension, Expect [LIB], automates interactive programs and simplifies regression testing. DajaGnu is an open source testing framework based on Expect [SAV].

Writing scripts is faster and easier than writing Java or C procedures. According to [OUS2], there's 5 to 10 times productivity gain on average by using scripting languages. Some of these gain came from the fact that variables can be used without first to declare them in Tcl/Tk. The fact that everything is considered a string in Tcl/Tk also facilitates gluing different programs together.

Here is a small example, just one line [OUS2] below, to give a flavor the power of Tcl/Tk:

button .b -text Hello! -font {Times 16} -command {puts hello}

The above line will create a button called Hello! in 16 points Times font and will print "hello" when clicked.

Another very powerful, general, and extensible scripting language is Python [LUT]. There is also no need to declare variables in Python. Functions, classes, and modules are all first-class objects in Python and can be passed or returned from functions. Note that many design patterns are greatly simplified in dynamic typing languages such as Python [NOR].

An version of Python implemented in Java is called Jython [PED]. Jython works especially well with Java classes. It uses many Java (bean) classes reflection properties to provide clever shortcuts to programming. It's kind of like a shorthand to writing in Java, providing a big productivity gain.

5.9 Stories in Leveraging Architecture Framework

Software is getting more complex and many specialty areas are emerging. One should avoid reinventing the wheel. Twenty years ago projects still wrote customized file systems or rudimentary database. Today very few projects will build their own DBMS. There are indicators that workflow systems are going through such a transition now [LEY], although many projects are still writing their own workflow sub-system. I have an unsuccessful story to tell here. A project I consulted on needs to inter-operate workflow from two systems. I am in favor to use a single (commercial) workflow engine to drive the workflow of both systems. However, for turf and political reasons, two identical workflow engines were used and communicate status and requests, not directly, but through application program interfaces. I pointed out that this approach would impose a severe performance penalty and error-prone as well. It's like writing your own workflow engine through application interface. But my warning was not heeded. A few months after I left the project , I heard that the project could only deliver a small portion of the features (and workflows) requested by the customers. May be the complexity of inter-working two workflow systems by application program interface contributed to some of the difficulties.

In other projects I worked on around mid-1990s, we had very positive experiences with Java Server Page (JSP) in easing the generation

of dynamic web pages. We also had good experience in using the factory method pattern to allow one of the system we built (which manages network cutover and the migration of network equipments) to manage many different type of network equipments with minimal code changes. We also had good experience in using the open source tool STRUTS framework for web-based front-end applications.

5.10 Useful Practices in Leveraging System Architecture Framework

I believe that reuse is the key to low cost and high quality and object- oriented approach is the best way to go because now reuse is possible at both the component and the design pattern level. One should also evaluate the possibility of assembling an architecture framework based on reusable components and design patterns appropriate for your problem domain. When appropriate, one should also use scripting language such as Tcl/Tk or Python/Jython.

Another important thing to emphasize is to design in flexibility and ease of maintenance. Assume that many features will be added later. Ask yourself what would be the impact to the system if various features are added or changed? Could one make the system "table-driven"? Could one leave hooks in for future expansion?

Another good practice is to have a good system architecture design review. Architecture de-

sign should not stop at the level of identifying hardware and software block diagrams. Real effort should be made to estimate system performance (throughput, response time), capacity (number of hits/minutes or users that can be supported), failure modes, recovery scenarios, scaling up strategy, security analysis, etc. In AT&T Bell Laboratory, one of the "Current Best Practices" is Software Architecture Validation. In this blue book, comprehensive review questions are listed in areas like performance, error recovery, and operation/administration/maintenance (OA&M). Projects for architecture review are asked to prepare answers for these questions. Typically a few experienced developers and software architects are asked to be the reviewers. The company realized that not that many people have software architecture experiences thus such resources should be pooled together for corporate-wide services. Such review points out potential problems and solutions of the proposed architecture to the project and provides excellent forum and learning opportunity for all project members.

My observation has been that frequently in a software system, the original architecture vision tends to get lost in time downstream in implementation. I think it would be very beneficial to have an architect appointed who will be seeing the project through to product delivery so that the design and implementation reflects, preserves, and realizes the architecture vision and goals the system started out with.

Software systems do not exist in a vacuum. Many systems need to interface with other systems. I think whenever possible one should use

open standards for interfaces (such as CORBA IDL, XML) and use standard tool packages whenever available (such as XML package in Java 2 platform).

5.11 Sharing and Project Evaluation

5.11.1 Sharing in System Architecture Framework

Two questions:

Most Useful - What have provided you the most leverage in the area of system architecture?

Toughest - What have been your toughest challenges in creating/integrating your overall system architecture?

Here are my inputs:

Most Useful - I have retired from active project management before the J2EE design pattern framework became available. I think today I would try to use that or something like that. In the past, I found Java remote method invocation (RMI) very useful for Java-based distributed systems and CORBA IDL for integrating more heterogeneous systems and environment.

Toughest - I have found that architecture issues to support rollback in a distributed environment very challenging. In telecommunication service provisioning, there is a so-called

"stacking orders" problem. Namely, various customer orders can depend upon and interfere with each other as they may reserve communication capacity on shared facilities. Since the orders need to take many steps to implement, and are subjected to customers' modification (change orders), it's quite a job to keep everything in sync and up to date, as the logic for changes and rollbacks could get very complicated.

5.11.2 Evaluation of System Architecture Framework

Evaluate to what extent your project has exploited opportunities in leveraging reusable components, design patterns, architecture framework, and problem domain scripting? Do you conduct thorough architecture review and addressed key architecture issues?

Exercise - contrast project with no leverage in component, design pattern or architecture framework reuse with one that maximizes the leverage. How to tell the differences between the two?

Chapter 6

Process Discipline

In [Yeh-4], the best organizations evolve toward perfection by the art of mastery. These organizations continue to improve and transform in multiple dimensions - passion to do one's best; learning, knowledge, and discipline; infrastructure, process, and simplification - and this is a never-ending journey.

In this chapter I will discuss a framework to manage and improve a specific software process, including techniques in data analysis, statistical process control and process modeling applied to software processes. In particular, I'll emphasize that, data collection and analysis techniques together with process improvement methods can go a long way to meet customer expectation and reduce cost of quality, whether statistical process control charts can be applied or not. I'll discuss the basis of statistical process control and point out that the typical metrics in software are not good candidates for those control charts. The concept of six sigma, a closely related topic, will also be discussed. However, certain software de-

fect removal processes, such as code inspection, system testing, do seem to be amenable to certain simple modeling. Assuming that those simple modeling works, I'll discuss how statistical process control then could be used to monitor the progress and quality level of those software processes.

As usual, provocative statements, examples, and useful practices are also included in this chapter.

6.1 Statements about Process Discipline

Evaluate the following statements. Do you agree or disagree and why?

1. Software is written by people. It is often very complex and highly variable. Hence it is not amenable to statistical analysis or process discipline.

2. I manage only one project. There's too little data for meaningful data analysis.

3. How to maintain the same process in spite of large turn over in team membership? What does it mean when one says there is a repeatable process? Does it mean the good process results can be repeated? Or merely that people will follow the procedures faithfully?

4. Standards stifle individual creativity.

5. Higher quality requires higher cost.

Here are my inputs:

1. **Statistical analysis could be applied to a phenomenon in spite of the fact that there are many variables and large variations as long as the variations are random. People's life habits are very different yet insurance companies can analyze life expectancy data and set life insurance rates. Statistics will not predict the outcome for an individual, but could be quite accurate to predict about the "average" for the population.**

2. **Depend on the size of an organization there may be many or just a few projects. Even within a single project, there could be many releases. Even within a single release, many task, such as doing system builds or regression testing, may need to be done many times. So I think there are always opportunities to apply some data analysis techniques to software processes in a project.**

3. **When there are many new people on the project, the process would surely be a little different even if everyone follows the same procedure. For one thing, people's skills are different. One could minimize the process degradation during personnel turnover through training, collection of process metrics, continuous process improvement. For those processes that can be controlled with statistical process control charts, there could be an unambiguous answer about process quality. Namely, one can monitor the process with control chart to see if the process**

is still in process control with the new team.

4. Standards versus creativity - one could argue that standards help one to make a better product without hinder creativity because one does not need to worry about the steps in doing something and can focus just on the solution itself. In addition, just about any solution can be constructed to follow the standard processes. The standards are focused on how to construct software, not the content of software construction.

5. Quality versus cost - one can argue that paying attention to quality in prevention reduces cost as cost of repairs is far more expensive.

6.2 Some Process Examples

6.2.1 Rational (now IBM) Unified Process (RUP) [RAT]

Rational is famous for its Unified Modeling Language (UML) standard. As one example of a more traditional software lifecycle, here is what Rational proposed for a unified six-step development process:

- business modeling
- requirement
- analysis and design
- implementation

- test
- deployment

Rational also recommends the following six best practices for software processes:

- develop software iteratively
- manage requirements
- use component-based architecture
- visually model software
- verify software quality
- control changes to software

6.2.2 eXtreme Programming(XP) [EXT], Is It for You?

XP has a number of spectacular success stories, however the approach is controversial in the industry. I think some of the XP practices - such as shared knowledge, uniform unit testing, frequent integration, monitor acceptance test results - all help to keep the discipline and mastery in place in software construction.

Here are the key XP rules:

- Written user stories;
- Simplicity;
- Frequent small releases;
- A stand-up meeting starts each day;

- Customer is always available;
- Code the unit test first;
- All code pass unit test;
- Integrate often;
- All product code is pair programmed;
- Collective code ownership;
- Re-factor whenever possible;
- Acceptance test are run often and score published;
- No overtime;

A closely related concept is called agile process [AGI]. It emphasizes that working software is the primary measure of progress and projects should deliver working software frequently and quickly.

6.2.3 Some CMM Key Process Areas

This is a good place to tie back the CMM framework with the five arts of the holistic Tao management framework we are using here. CMM also identified key practices to support each of the CMM key practice areas [CMM-2]. CMM key practice areas for each management art are:

Process concerns - organization process definition, quantitative process management.

People concerns - training program, inter-group coordination.

Timing concerns - configuration management.

Leverage concerns - technology change management.

Vision concerns - I do not find any that fit with this art (vision).

There are also many CMM key practice areas that are more like specific individual process, such as requirement management, project planning, project tracking and oversight, quality assurance, defect prevention, and subcontract management. These processes could be improved by applying the six-part framework, to be discussed next, to them.

6.3 How to Improve Process Quality

6.3.1 A Six-part Framework [YEH] to Understand a Software Process

CMM framework provides an overall roadmap to improve the software manufacturing capability for an organization. However, one still needs to know what to do specifically with a specific process. The following six-part framework focuses instead on how to understand a specific software process, such as system requirement process, or system build process, or code inspection process, etc. Here are the six steps:

Define the process - What is exactly the process? What does quality mean for this process?

Measure the process - What are the metrics [GRA] for the process? Are they related to the product metrics in any way? How does one measure size, defect, or effort in software?

Analyze the process - What are some useful techniques one can apply to analyze this process?

Monitor and control the process - Are process control charts [GRA-2] for hardware quality applicable here? What would be the meaning of tolerance in software?

Modeling the process - Could some processes amenable to modeling [MUS]? Could one apply these models for planning, estimation or prediction?

Improving the process - Could techniques for continuous process improvement such as root-cause analysis, post-project reviews, be applied here?

In what follows I shall touch upon what is quality? and a number of data analysis techniques from this framework. The topics of statistical process control and how to apply them to software are big topics and will be discussed in separate sections in the following.

6.3.2 What Is Quality?

For a product to meet users' expectation, one needs to know what are the expectation and what

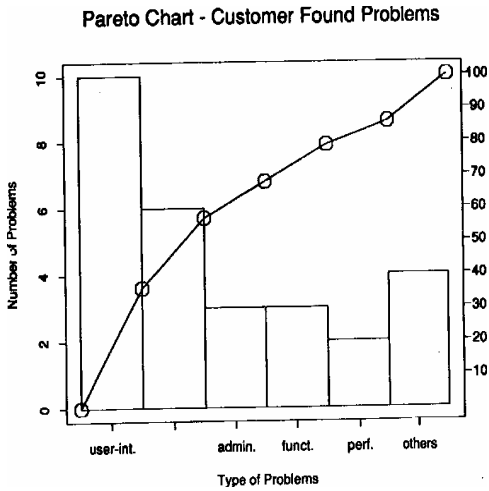
are important to the customers. So it's very important to get input and feedback from customers and users. Once one knows what is important, one can set standards and set metrics and measure the metrics to monitor the level of conformance and quality. Here are some factors for quality that are typically important from a customer or user perspective:

- Fitness of use
- Conform to specification
- Low cost
- Reliability, last a long time
- High performance, high throughput, fast response time
- Ease of use, friendly user interface
- Minimize variance

6.3.3 Some Data Analysis Techniques Applicable to Software Processes

Benchmarking Once the metrics for a process has been defined, one can keep track of it and review improvements over time or compare results with other similar projects. It's most important to know what is important for the customers. In software, typically cost, quality, response time, performance, feature-rich are all important. To monitor these important metrics and to see if you are improving and meeting your customers' expectation is to begin doing quality management of your products and processes.

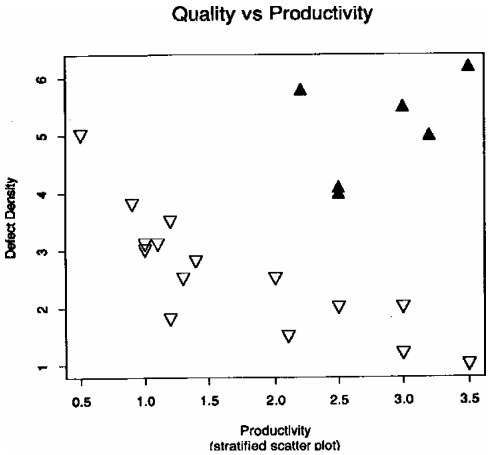
Pareto analysis Frequently attributes or properties distribute in a very non-uniform manner. For example, in the figure attached, 60% of defects in software may be concentrated on user-interface and reliability problems. Pareto analysis is a charting technique to identify the leading contributors to a given metric. To analyze your cost, defects, response time and other key metrics by Pareto analysis, such as by stage of production, is to begin to zero in on what are the biggest opportunities for improvement.



Trending Keeping track of how metrics change with time is important. One can tell from such tracking whether process is getting better or worse.

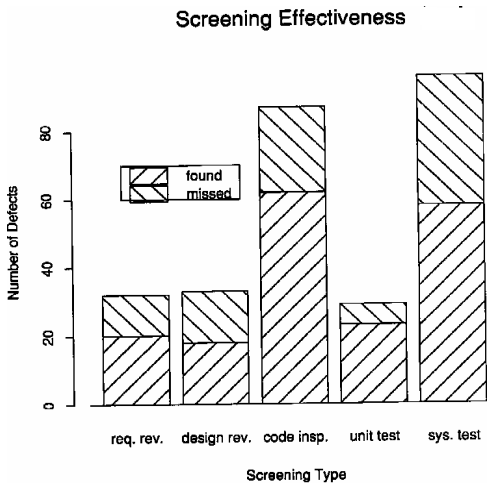
Scatter plot This is a very useful technique to study if there are any relationship between two

variables among the population of data measurements, such as functional points versus effort (person-hours), defective density versus productivity. If there seems to be a nearly linear relation, one could do regression analysis to fit the data points into a linear curve. The equation could then be used as a basis for estimation in future projects. Sometimes, the data may fall into multiple sub- populations (stratified scatter plot). For example, in the figure attached, quality in some projects goes down when productivity goes up, but in other projects quality and productivity go together. Then it would be very interesting to find out more details about the differences and practices between these two groups of projects.



Decomposition Sometime it is useful to peel the next layer of onion and decompose the data into finer details. One example given in the attached figure here is not only to look at the screening

efficiency of the overall software production process before product reaches customers, but also to look at the screening efficiency of each software development lifecycle phase. Namely, what percentages of defects were found right at the phases where the defects were introduced? If many defect were introduced earlier on in the lifecycle but were only found later, that would be an opportunity of process improvement as cost of detecting and fixing defects goes up quickly in later lifecycle phases.



6.4 Statistical Process Control for Software Processes

6.4.1 Control Chart Concepts [GRA-2]

In hardware manufacturing, one could use process control chart to monitor if a process is in statistical process control or not. Let us review

here what are the meaning of these concepts and whether they are applicable to software construction.

We probably all have seen a normal distribution, or the curve that shapes like a bell. In statistical analysis we find that if there are many random factors that affect the value of an attribute, such as the length of a nail made by a machine tool, then the distribution of the length tends to be a bell shape curve or a normal distribution when one plots over large number of samples [BOX]. Most of the data will cluster around the mean value. A measure of how broad or narrow of the bell shape curve is given by the standard deviation (sigma), which is the root-mean-square value of the difference between the sample data from the mean. For normal distribution, about 68% of data are within one standard deviation (one sigma) from the center (mean), 95% within two standard deviation (two sigma) and over 99.7% within three standard deviations (three sigma). Thus it is very unlikely that a data point could lie outside the three standard deviation unless the process is abnormal or out of control in some way.

It is expected that the points collected in a random sequence of sampling would tend to jump around the centerline (mean value) randomly and fits well with a normal distribution. By plotting the data, typically averaged over a subgroup of four or five samples from the same batch [GRA-2], one could tell if the process has changed from batch to batch and whether the process remain in statistical process control. Such control charts, called X-bar and R chart, when applicable, pro-

vides an easy way to tell if some non-random causes is present to cause the process to be outside the (three sigma) control limits. If it is outside control limites, then one can find out the causes and fix the problem. The X-bar represents the average of the subgroup. The R stands for range or difference between the largest and smallest value in the subgroup. By using average of ranges in these subgroups, one can compute the upper and lower control limits without calculating the more complicated standard deviation [GRA-2].

One might wonder why we need to do average in a subgroup, instead of using the value of each individual nail? There is an important reason. If we know for sure that the population where we draw our samples from are distributed in a normal distribution, then we could use the individual sample as a basis to calculate the control limits and control charts. But in many cases, we don't know that or the population actually is not distributed in a normal distribution. For those cases, use subgroup average instead can save the day.

There's an important theorem in statistics, called the central limit theorem. Basically, it says that the average over an sample (such as the average length of five nails, or the average over the face values of throwing ten dices) tends to approach a normal distribution when the number within sample gets large. This is true even if the samples are drawn from a population that does not distribute normally. This is because random errors from different factors tend to cancel out.

To illustrate this, let us consider throwing dices. If one throws a single dice, the score is

very discrete, either a 1, 2, 3, 4, 5, or 6. (So here, the population is not a normal distribution at all.) Each value will have 1/6 chance to appear. But for throwing two dices, the average probability will no longer be 1/6. The chance of a 1 or a 6 will be only 1/36, when both dices are all 1 or all 6. But there will be more probability for a 3.5 value because one can get a 3.5 average value by having a (1,6) or (2, 5) or (3,4) combination. But throwing 5 dices together and take the average score, you got a lot more chances to get an average around 3 or 4 than a 1 or 6. Because for it to get an average score of 1 or 6 you need to have all five dices to be 1 or 6. But to get a value between 3 and 4, there are many more combinations. By the time we throw ten dices, the probability distribution will look very much like a normal distribution [BOX]. Thus it is important to take the sample average. Typically sample size of four or five is used and sufficient to get a normal distribution for the subgroup averages.

There are other types of control charts in hardware manufacturing quality control such a p-chart. It is based upon binomial distribution and has to do with the probability or percentage of defective units in a batch. I am not aware of application p-chart for software defect analysis. May be it is hard to decide what is the equivalent of a batch or a unit in software. Would a thousand lines of code or a module be considered a unit?

The fact that process is in statistical process control does not imply that the products produced (such as length of a nail) would meet customer's specification. For the latter, it is called

tolerance or the range of variation acceptable to the customers. Customers may require tolerance limits much narrower than the control limits of the process. If so, even if the process is in process control, many units of the products will still be rejected. In general it is preferable to have tighter (that is, very narrow) control of the process, with control limits (three or more sigma) well within the tolerance limits. If that is the case, may be one parts could be used for many products. Cost of quality is greatly reduced, as there's little rejection or repair. Hence the push for six sigma which I shall discuss next.

6.4.2 Six Sigma Concept [SIX]

Six sigma is widely deployed and credited with helping companies to make huge savings. In addition to pushing toward very low levels of defects as a goal (six sigma or six standard deviation represents defect level at a few parts per million). In six sigma, the process control needs to be so tight that even if the product is six standard deviation away from the mean, it still is within tolerance set by the customers! Six sigma movement also emphasizes quality improvement and specific roles in quality improvement for individuals. Recently, there's a lot of interests to apply six sigma concept to software projects [SOF].

Six sigma covers most of the same steps in the six-step software process framework mentioned earlier, except for the modeling piece. But modeling is really crucial to make some software processes amenable to statistical process control, which in turn is the basis for standard deviation

or six sigma concept to be applicable to software processes in the first place. There are two different improvement models in six sigma, one for existing process, and one for new process. The steps to improve existing process, DMAIC, is consisted of: define, measure, analyze, improve and control. The steps to design new process, DMADV, is consisted of: define, measure, analyze, design, verify. Note that neither mentioned modeling.

Here are some roles important in the six sigma approach - green belt (beginner), black belt (expert), master black belt, process owner, quality champions.

6.4.3 Can One Apply Process Control Charts to Software Manufacturing?

Can control charts be applied to software manufacturing processes? Hardware manufacturing processes aim at producing some widget with some properties, such as the dimension, or the composition of a widget, to be precisely controlled within limits. What is the equivalent of that for software? Metrics important to software include things like quality, productivity, customer satisfaction, etc. However, since there is no particular reason to produce software at a particular rate (the faster the better) or with a particular level of defects (the fewer the better), one would not expect that software productivity (as measured by lines of code or function points per person-hour) or defect density data to follow a normal distribution. In fact, we like these metrics to be

as low as possible (zero for defects) or as high as possible (productivity).

Furthermore, the units produced (whether lines, or function points, or modules) are really very different from each other. In contrast, the units produced in hardware manufacturing lines are all the same. Another difference is that each unit produced in software is not the end product itself. The end product, the software system, needs to integrate all the units and use them together. So the p-chart mentioned above is also not applicable in a software situation.

So in general I think to focus on statistical process control or getting process to have very narrow standard deviation (high sigma) is the wrong focus. Instead, I think one should focus on what's important to customers (customer metrics) and apply data analysis techniques to improve on these key metrics. These process improvement can still be carried out even without process control charts.

However, there are software processes where process control charts have been applied successfully. They are all related to the processes of software defect removal. The crucial differences for these processes from other software processes is that these processes frequently obey simple models under certain conditions. More details will be given in the next section.

6.5 Modeling in Software and Statistical Process Control

6.5.1 A Simple Software Reliability Model

It was observed that frequently defect removal rate in system testing tends to be proportional to number of defects remain in the software. In other words, history has no bearing about the future, defect removal is a random process that has no memory and does not depend on what had happened in the past. For such testing processes, it's easy to show that the system failure rate, when plotted on semi-log scale, would decrease linearly with the cumulative testing duration. As defects were found and removed from the system, one expect to encounter lower rate of failures. For those processes, one could expect a certain level of failure discovery during a certain duration of testing, thus the testing process could be under statistical process control to remove defects at a controlled rate. Thus for those processes one expect the defect removal rate to fluctuate around the line predicted by the model, and a control chart could be constructed to monitor the effectiveness of testing. Let me introduce a little mathematics about this model.

If we let f_o be the initial number of defects in the system, and f be the cumulative defects found at time t, then $(f_o - f)$ represents the number of defects remain in the system. A very simple software reliability model assumes that the following

for the rate of finding and fixing defects,

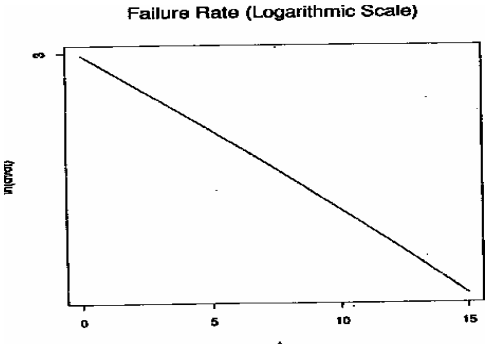
$$\frac{d(f_o - f)}{dt} = \frac{-(f_o - f)}{T} \tag{6.1}$$

In other words, the rate is proportional to the defects remain in the system. T is a characteristic time period. It turns out that this model predicts that

$$f = f_o(1 - e^{\frac{-t}{T}}), \tag{6.2}$$

$Tln2$ represents the time period to cut the remaining defects by half, or the "half-life" in defect removal. $Ln2$ represents natural logarithm of 2.

According to this model, if we plot the defect removal rate on semi-log scale with time t, it would be a straight line. A good example of this model working is given in G. Kruger's paper in Hewlett-Packard Journal, June 1988. The failure rate data fits well with the model and one can use the model to predict and set the time for release to production, once the software hits



the target of software quality desired. Such technique is also useful to estimate the effort level or duration needed to complete testing.

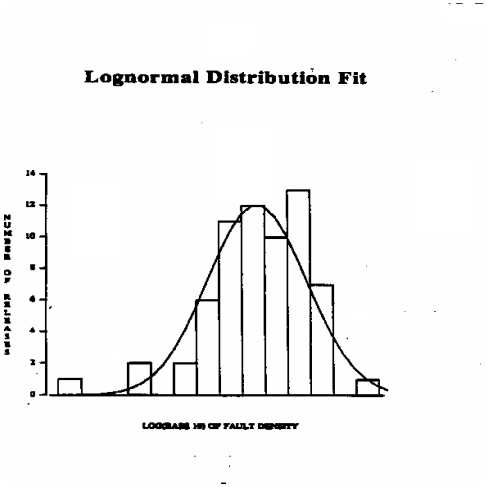
Another application of the same model by John Musa is the control chart used to monitor the progress of testing for a system T1 [MUS]. The control chart was constructed using the data from the first 100 failures. Subsequent data conforms well with the prediction of the model and upper and lower control limits. Thus here is a good example on how to use statistical process control chart to monitor the progress of testing to reach a desirable level of quality in product.

In John Musa's software reliability modeling, it's important to set up the system testing profile (the "operation profile") to simulate how users will actually use the product in the field, both in system load and type of usage, so that there's no change in failure discovery pattern when the products are released to the field. It's under these type of testing conditions that the simple reliability model seems to work best.

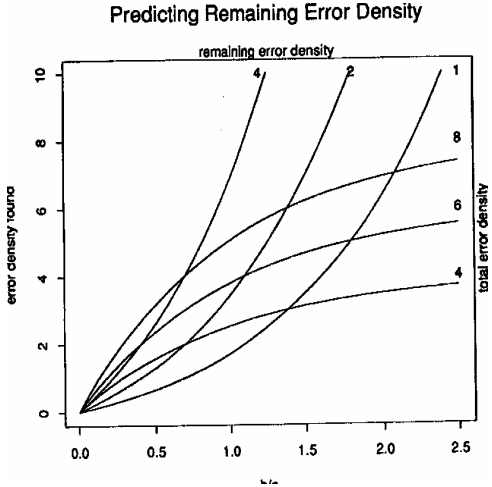
6.5.2 Lognormal Distribution for Data from Groups of Projects

The next application is not so much about process control chart but about whether one can use a normal distribution. When there is corporate-wide effort to improve software quality, there will be quality metrics data from many projects. As time goes on, one expects the quality of most projects to improve. A natural question is to ask how should one gauge the progress of such collec-

tion of projects? I was involved with just such an analysis at AT&T Bell Laboratories some years ago and have reported my finding in a conference [YEH-3]. The main conclusion is that lognormal transform seems to be a good model to use to monitor the overall progress of quality improvement of projects. The quality range tends to be wide but could fit with a lognormal distribution well (see figure attached). Quality improvement with a sharp boundary (zero defects) implies that improvement can not be expected on an absolute magnitude scale, but could be expected on ratios. Namely, every project could be expected to, say, cut its defect density level by half over a certain period. It seems reasonable that as defect density level gets lower and lower, it would become harder and harder to reduce it further. So it would not be reasonable to ask all projects to reduce defects by, say 1 defect per thousand lines of code, in a year.



6.5.3 Model for Code Inspection Effectiveness [CHR]



Yet another example of the use of modeling is the single parameter model for code inspection effectiveness. In this model, the fraction of defects found at inspection, f, is related to the inspection effort level h (in hours per thousands of non-comment source lines), by

$$f = 1 - e^{\frac{-h}{n}}, \tag{6.3}$$

where n represents some characteristic inspection effort level, like the T in the previous model.

With this model, one could find a family of curves (see figure attached) that relate the total error density, error density found, and remaining error density with the effort level. Thus by knowing the error density found and effort level applied, one could use the model to estimate the remaining error density in the system. What is interesting is that by analyzing the actual data the

researchers concluded that there were great potentials in finding more defects at code inspection by increasing effort level by a modest amount, and they were proven correct once the projects adopted their recommendations.

6.6 Stories about Process Discipline

I already mentioned earlier that a good rule of thumb in software quality improvement is to ask for improvement in ratio or percentage in defect reduction, not by reduction in absolute magnitude of defect levels. If current level is 4 defects/KNCSL, one may ask for a target of 50% reduction, but not by reducing by 2 defects/KNCSL. (KNCSL stands of thousand not-comment source lines.) The reason is that for projects with defect level at 2 defects/KNCSL already, they need to reduce to zero defect to meet that target, which is not realistic.

There was a project I did internal consulting work on. The testing period was very tight (2 weeks) and people working very hard, but can't get the system to stabilize. They seemed to find as many defects as they did previously and could not even run through all their test cases. It turned out that there was too much churning for testing and development, so new problems were introduced as fixes were applied. By extending testing cycles to four weeks and set up tighter control on entry criteria to system testing (clean build and pass all basic feather tests), the project

was able to stabilize the load, and got the full benefits of all test cases to get rid of bugs.

I had worked on a network management system project for a number of years. There were a good number of personnel turnover during that time. What helped to bring new people on board quickly included extensive project library, training sessions, and mentor assignments. These steps helped new people to become familiar with both the processes and the product, and there was always someone to help them when they have questions about "how we do things here?".

6.7 Useful Practices in Process Discipline

The five arts of business provides five dimensions to tune a project. It's an iterative process depends on project's situation and opportunities. Process improvement is a never-ending journey. There are many ways to gather lessons from a project, such as root-cause analysis, post-project review, cause-effect diagram, control charts, customer feedback, suggestion box. All could help to reduce the cost of bad quality. A most important aspect about learning lessons is that lessons learned must be translated into process changes, so we don't repeat the same mistakes in the future.

Process disciplines are in managing a lot of details - configuration, translation from user scenarios to requirement modeling, from model to design, from design to code, and manage the state of

the code in unit testing, integration testing, system testing, etc. All these steps need to be done reliably and repeatedly.

High quality in end product depends on good control of processes. Thus it is important to collect process data, analyze them, and use the results for process control.

Finding defects in the field by customers would be too late. It is best to prevent defects from being introduced. If defects were introduced, it is much cheaper to remove them in earlier stages of software production.

Process can be improved continuously. Root-cause analysis and post- project reviews are good mechanism to uncover lessons learned. It is important to translate project lessons into process changes and improvements so that same mistakes will not happen again.

Software process relies a lot on people following procedures and supporting the process guidelines. Thus training and in-process metrics collection are very important to keep the process functioning well. Automation and tools also help a lot.

Don't compare apples with oranges. Don't use process data (such as defect density from code inspection) to punish people. Don't give people a reason to play games with productivity or quality metrics data. Compare and improve on your own projects, such as to seek improvement in subsequent releases.

Maintain the process discipline and mastery. Consider some of the methods from XP, such as total unit testing and frequent regression/integration testing to ensure code base was

not broken by new or changed code.

6.8 Sharing and Project Evaluation

6.8.1 Sharing about Process

Here are two questions about sharing experiences:

Most Useful - What have you found to be most useful to achieve a stable, repeatable process for your projects?

Toughest - What have been your toughest challenges in the area of having a stable, repeatable process for your projects?

Here are my inputs -

Most Useful - Management support and process training and product documentation are all very important. Process related activities do add more work to project members. People need to understand why we need to do it and it is important to get people's buy-in about process quality monitoring. Sometimes people are nervous to report defects in other's code (such as during code inspection), as not everyone could be an "egoless programmer". Reporting only summary results - with no analysis on individual's defect rate and clear signal from management not to tie these data to performance would help to dispel the fear. There is no basis

to tie where defects show up with the individual writing the code. The defect may be seeded in requirement or design. In any case, it's not productive to trace defects to individuals. Aside from project members' buy-in, training on process is also very important. Data collection and analysis is also very important, so one knows how the various processes are working, where the effort were spent, and how buggy is various lifecycle phases, etc. These in some sense represent the true capability of the team.

Toughest - Any changes, personnel, new development environment or tools, could impact the process. This is especially true for personnel turnover. It's very tough when there is large turn over. (When a project is perceived to move into maintenance mode, many of the more experienced developers may decide to move on.) It will take a while for new people to get into the swing of things, so good documentation and training are very important. Another challenge is to adapt local processes to organizational standards. These standards have a tendency to change from year to year with a new CIO or IT leader, although the essence could very well remain the same.

6.8.2 Process Evaluation

Evaluate process discipline in your project. Do we know what's important to the customers? Are these metrics measured and meet target? Are

processes in control and meet tolerance limits? Are we doing process improvement based on data analysis? What works to ensure new people will get the disciplined way? What needs be improved?

Exercise - Contrast project without process discipline with one which has good discipline. How to tell the differences?

Chapter 7

People Development and Team Building

Many roles of leaders are discussed in [Yeh-4]. These roles include visionary/architect, teacher/coach, and steward. We already discussed the role of leader as visionary for projects in Chapter 2. In this chapter, I shall focus on some techniques for leaders to be a teacher and coach, such as techniques in trust building, people development, negotiation and team building. As usual, stories and useful practices related to management principles here will be included.

7.1 Some Statements Related to People Development and Team Building

Here are some statements related to people development or team building. Ask yourself whether you agree or disagree with the state-

ments and what are your reasons.

1. To negotiate for best results, information should be shared only on a as needed basis. Don't ever tell people on your project your bottom line.

2. Be nice to your people and they will be nice to your projects. Best way to take care of your business is to take care of your customers. Best way to take care of your customers is to take care of your people.

3. In a tough assignment, one can only win by demanding the best from everyone on the team. This means that there is no room for softy, wimpy niceties. Life is tough and tough situations require us to be tough with each other.

4. People are our most important assets. One should try to cultivate and develop them.

5. People is just one component of production. Just like other production components, such as capital, plant facilities or other resources, people should be used to the maximum extent for business profits and replaced when exhausted or outdated.

Before I discuss the above statements, I need to point out that my view and the techniques I recommended in this chapter is based on the belief that win-win is a much better approach in working with people on projects. This is because the goals of the project manager and project

members should ideally be aligned for best results. A good way to do that is for the project manager to always keep the best interests of all project members in mind. I should also point out that being "nice" does not mean one has to be "soft". I try to be nice to people but have no problem to provide negative feedback to team members or even to put them on probation if I think it's necessary.

Obviously, win-win is not the only way to achieve good outcome in negotiation. Fear could be a very strong motivator, especially at times when good jobs are hard to find. There are managers who believe in "speak softly but always carry a big stick." But using fear would fundamentally change the values of the company and both the work environment and the relationship between people will suffer as a result. Job security is always a big concern for people who needs to work for a living. I think it's seldom necessary to remind people their job is on the line if they don't perform. The people I work with almost 100% want to do a good job. I think a big part of manager's job is to remove roadblocks so they could do a good job.

Another management philosophy (also practiced by many parents) is to set impossible goals in the belief that this is the way to get superior performance out of subordinates. Since people will fail, the thinking goes that one might as well set a very high bar so that even in failure the result might be acceptable. Team members are always reprimanded that job results are not "good enough". For me, I like to be their coach and cheer leader instead. I like to accentuate the pos-

itive and applaud their strengths instead. Set people up for failures could be very frustrating for both the manager and the subordinates.

Still another management philosophy is to disregard people's needs, feeling or careers. Everything is secondary and can be sacrificed for the sake of project success. The manager may be achieving miraculous results but left a string of dead bodies in his/her wake. It is a ruthless approach that would just use people up, "take them in and spit them out", just like raw materials or other disposables. I am in favor of a warm, caring, respectful approach to people, even if job security can not always be guaranteed. After all, if one asks oneself, who wants to work for a heartless, ruthless manager over the long term?

With the above caveats, here is my take to the above statements:

1. I disagree with the first statement. People need to know where they stand and what would happen if there is downsizing. Project manager should try to build trust, team spirit and goal alignment. Communication should be open, honest and frequent. People should feel valued, respected, nourished, not squeezed to the brink of burn-out, used and dumped. They should be the first to know directly from their managers if some major decisions affect them. In a project, managers and project members are in it together. Their interests should be aligned, not against each other.

2. I agree with the second statement. Happy workers do make happy customers. Happy

customers will bring good business and make happy company owners.

3. I disagree with the third statement. Tough situations do not automatically imply that one should get tough with each other. In fact, one can argue that because the situation is tough, team members need to support each other even more, as we are all in this together.

4. I agree with the fourth statement. It is people that makes all the differences for project in most cases.

5. I disagree with the fifth statement. People should always be respected, including contractors and temporary help working on the project. Outsourcing or using contractors are certainly important options to keep software development cost down. Expectation and relation with outside contractors is indeed different from that with long term employees. However, one can still be nice to contractors even if the contract indicates clearly that it would be a short-term assignment and contractor assignment could be terminated with very short notice. Contractors will still respond differently if the manager shows caring and tries to make every project member's life more pleasant, including contractors. It's part of human nature, we respond nicely to people who treat us nicely.

Another bias of mine in people development is this: I firmly believe that it's very important

to accentuate the positive in people development. If one looks for it, everyone has some talent and strength. I would focus on that and help the person to shine in some way. If one focuses only on addressing shortcomings, I am afraid that most people have so many areas to improve that the person will easily get discouraged. Even if the person has improved on most areas, if there's nothing to stand out, s/he is still only a so-so performer. Time and again I have obtained good results by helping people to discover and develop the talents and strength they already have.

7.2 People Motivation

What motivates people in the work environment? Income and job security is certainly the number one motivator. It is the main reason why people work for pay in the first place. It will be hard to motivate someone who believes that he or she is grossly underpaid. In today's environment managers can no longer guarantee jobs, but one can still help workers to grow and be more marketable.

Assume that pay and benefits are not a problem, then other factors could also be very important. Software technology is evolving very quickly, so to keep up with new technology is very important for developers to remain marketable. This is especially important for people in today's environment as people find out that very few companies now can provide life-time job security. If job assignment could provide opportunity for growth and skill development, it would

usually be a big plus.

People usually look for recognition. If a person has a great skill in some areas, it would be nice to take advantage of that strength. Help people to shine and recognize their achievements. People like to feel special. They like to do good work and be recognized for it.

People spend so much time at work so the work camaraderie becomes very important to them. Provide a good environment and build great team spirit would certainly help to attract great people and help keep them there on your project.

Last but certainly not least is that people appreciate someone who could listen and care about them. People need to feel valued and cherished. It is part of our human nature. Spend time one-on-one with each member on your project and just listen and pay attention to what each has on his or her mind.

People's overall needs are very similar but do differ depend on their different life stage or development stage. A young person just starts out on a career has very different needs from someone who is near retirement. They may need different help from the manager. Best way to find out is to ask them.

To start a new project involves many things for the project manager. One important task is to get to know the project members and to build mutual trust. I shall talk about trust building techniques next. Good human relations depend a great deal on give and take. Take care of your people as you rely on them totally to take care of your projects. Grumpy workers can give customers a hard time and drive them away.

Managers automatically know it is very important to have a good relationship with the boss. It is also very important to have good relation with one's peers, as no project is standing by itself. One always needs to interact with other projects, as suppliers, consumers or other type of system to system interaction. Build trust with your direct-report and team members. Its awfully hard and tiring to need to police everyone to do a good job. Team members and peers are not our adversaries. We are all in the same boat!

Take care of your people and they will take care of your projects. One can be nice in many ways. You may not be able to guarantee jobs or give salary raises, but you could still help workers grow and be more marketable.

7.3 Trust and Relationship Building [FIS]

Roger Fisher and Scott Brown of Harvard Negotiation Project have produced an excellent book on how to build trust and relationship. They recommend applying their techniques to ALL relationship, including adversary ones. I think the techniques certainly should work well for people within a project, as there is already a natural basis for alignment - for the success of the project. Looking for goals both parties can support and emphasize common goals is a very important step in reaching agreements. Remind people that "we are in this together".

Here are the six principles from "Getting To-

gether – Building Relationships as We Negotiate" on how to build trust and good working relationship:

Understanding - Learn how the other party see things and what is important to them.

Communicate - Always consult before decide on things that affect the other party; always avoid surprises.

Be Reliable - While not necessarily wholly trusting the other party initially, be wholly trustworthy yourself.

Acceptance - Deal seriously with others even if you strongly disagree with some of their views.

Use persuasion - Never use coercion, always try to reason and find common goals and common grounds. This is especially important when one works with people with less power than you, such as with subordinates. Power has a tendency to coerce and corrupt the relationship.

Balance emotion with reason - Good relation and communication requires one to pay attention to both reasoning as well as to any emotion or feeling of your partners.

7.4 Approach to Negotiation [FIS-2]

Also from Harvard Negotiation Project, Roger Fisher, William Ury and Bruce Patton wrote "Getting to Yes – Negotiating Agreement Without Giving In" to guide people on win-win negotiation techniques. There are four major principles –

Separate people from the problem - Don't attack a person just because you and the person disagree about some issues.

Focus on interests, not positions - Don't lock into a bottom line position. First explore concerns and needs.

Invent options for mutual gain - Instead of fighting for a bigger slice of the pie, see if both can have more by having a bigger pie or change the scope of the negotiation in some ways.

Insist on using objective criteria - If one can't get agreement or proceed with negotiation of the issues at hand, may be one can negotiate about some objective criteria to guide a fair decision or negotiate a common ground or procedure on how to move forward.

7.5 Ideas on Team Building

There are many good techniques about team building. Here are some that have worked for me –

Get to know each other Get team members at the beginning of project to sit down for some team building sessions. Get people to introduce themselves. Share goals of the project. Identify people's concerns about the project. Find out what do they like to see happen or afraid would happen.

Build alignment Explain the importance of the project, how does it support corporate business needs, customer needs and the vision of the company. Through one-on-one session and small group meetings help the individuals and subteams to align their goals with the overall project goals. Point out how the project tasks could help them to achieve their personal goals.

Empowerment Empower the team or sub-team to be able to make many decisions about their work, including the roles team members can assume. In order for the team to assume more responsibilities in seeing tasks completed more quickly with less time and cost, team members need to have more autonomy to move quickly to solve problems. Always be there to support and jump in to help when needed. A good rule to win team's support is always be there in the trenches with your team when there are hardships like overtime. Lead by example.

Matching talents Select people with compatible temperament and complementary skills for team. People with vastly different skill level are not a good mix unless it is made clear the two have

very different roles or that one serves as mentor for the other.

Create good times Team building is also very much a matter of sharing good times, including informal time together such as lunch breaks, luncheons.

Celebrate successes Don't wait till the end to celebrate success or to recognize team and individual's special contribution. Find many opportunities to provide positive feedbacks and celebrate small wins.

7.6 A Story about Self-Managed Team [YEH-2]

I used to have the development responsibility of a large network management project. There were a lot of problems between the three disciplines - system engineering, development and system testing. As a result, the production cycle for a new release was very long (16 months), and the product was very buggy. People worked very hard and were pretty unhappy. Even simple problems took a long time to fix. Customers were very unhappy.

I was still fairly new to the project but decided that we need to break up the barrier of interfaces between disciplines. It just seemed very inefficient that one needed to escalate and went through managers in order to resolve technical problems. People who should share a com-

mon goal to meet customer needs were more like adversaries and tried to find blames with each other.

I got the buy-in from my boss (who was the overall project manager) and peers to try a very different approach. They also recognized that the current way in running the project was not working and were just as frustrated as I was.

Under the new approach, we reorganized the people on the project into various cross- functional teams. They were small (3-5 people) and responsible for end- to-end result of a feature. Features were decoupled as much as possible. As a result interfaces across the discipline for that feature would now be within that team. To support them, they were empowered to be flexible on interfaces and roles, so they could deliver quickly and with high quality. To support this new venture, we also bent our process a little, to allow more overlap between architecture and system requirement phases, instead of strict waterfall model for product development.

Management support was very important, so people can stop playing the blame game and get on to talk with their teammates and get the job done. The results had been very gratifying. We started to be able to deliver small enhancements very quickly, which used to be a big sore point with our customers. We achieved 25% cycle time reduction to 12 months. (Certainly sounds awfully long in this internet-time development of today, but was a big improvement then, 1990). More importantly, the quality of our product went way up. Number of serious defects were reduced by an order of magnitude. Customers

no longer needed to do debugging for us. Empowerment was also very important. There were much role stretching. People took on additional roles to help each other out and to achieve the goals they set for themselves. They were happy with the outcome and they were surprised that they had hidden talents they don't know about. It was a lot more fun to get things done well and celebrate successes instead of being paralyzed by infighting and blame.

7.7 Useful Practices in People Development and Team Building

Once you find good people, take good care of them. Meet with them one-on-one periodically to find out if something is bothering them. Address their problems. Help them to develop their careers. Help them to find out their strength, and give them opportunities to show case their abilities. I am a believer of focusing on and developing people's strength. I am not saying that people's weakness should be ignored. But I think people can be more successful by focusing on their strength.

Collect feedbacks and try to improve based on input. Eliminate fear. Collect anonymous upward feedbacks as well as feedback from peers, customers, boss and other stakeholders about you.

Trust-building and win-win negotiation. Follow the trust and relationship building principles for all work relationships. Follow the win-win negotiation techniques for all negotiations.

Give credits to your people. Develop and showcase talents and success. Give them the opportunities to shine and celebrate success. The overall project's success is already your success, so project managers do not need to claim all the credits over their people.

Treat people fairly. Don't play favorites. Don't give choice assignments or hardship assignments to just a few. I try to balance more interesting work (new technology or fun job) with hardship (maintenance, take beeper and on call for field support).

Match talents and temperament in setting up teams. Also give the team flexibility and leeway to make decisions and to get things done.

Nothing beats finding good people to begin with. If one can hire, recruit carefully. Degrees from good schools is certainly one level of certification. But many software engineering skills are not covered that well in school, so industrial certification is a very useful supplement. Good recommendation from previous project managers are also important.

One likes to stretch the people but not too much. A job too easy is boring. A job too hard and

people will feel defeated. A good job design seeks a good balance between the job challenges and person's skill level.

Remember managers roles. Remember managers role as leader - visionary/architect (for the project/team), teacher/coach (for people/team), and steward (to build up project assets, core competency, etc.)

7.8 Sharing and Project Evaluation

7.8.1 Sharing about People and Team Development

As usual, I like to ask the following sharing questions:

Most Helpful - What have been the most useful approaches for you to achieve happy workers and strong team?

Toughest - What have been your toughest challenges in the area of people and team?

Here are some input from me -

Most Helpful - Help people learn. Over the past twenty years, I found that good people like to learn new things and there are always many new technology one can learn in the software business. I have been very fortunate that during my time as project manager I have resources within my discretion to

help people with this learning, such as training and books, as well as applying their new learning in their assignments, as the projects I worked on were very much at the leading edge of technology.

Toughest - The most challenging situations are definitely sudden downsizing. I need to place a large number of project members very quickly. With help from my manager who had many good contacts, we were able to place our employees as a block to another project, so no one was left hanging. I was also able to help most of the contractors to find work elsewhere. The other tough situation was to follow corporate guideline and to ask contractors to roll back their hourly rate. Now a day a thing like that is getting more common. But for the first time, it was very tough to tell people we need to cut their salary as people's ego are very much tied up with the money they can earn. While team members understand that it was not my fault, I still found it hard to ask people to cut pay. (I certainly would feel very bad if someone wants to cut my pay, as one needs to make a lot of adjustments).

7.8.2 Evaluation of People Harmony in Projects

- Exercise - Evaluate where your project is on people development and team building. What works? What areas need improvement?

Some questions to ask: Are people pretty happy, their needs met? Is team members work well together, team spirit high? Does management and subordinates trust each other? Are project members skillful in win-win negotiation and trust building?

- Exercise - Contrast project without teamwork and alignment with those that maximize it - how to tell the differences?

7.9 Technical Management - Opportunities for Action

Here we come to the end of this book. I hope that I have illustrated how to apply the five arts of business management to software project management. I hope that some of the points in this book are helpful for your project management issues. Here's a summary of key points I like to reinforce:

- Align project focus with corporate vision, get project members to understand and support the vision/values
- Minimize risks - simplify, automate, standardize, rapid prototype - for dependable and sustainable project successes.
- Leverage open standards and Company resources to simplify and standardize the software manufacturing environment.

- **Leverage stable version of open source software to assemble and automate various aspect of the software manufacturing environment.**
- **Leverage experienced software architects to conduct system architecture review and address key architecture issues.**
- **Leverage component-based approach to reduce cost and increase productivity for software product.**
- **Leverage design pattern and assemble patterns into architecture framework, such as J2EE, .NET or MVC/STRUTS for web applications, to reduce cost and increase productivity.**
- **Leverage powerful scripting language, such as tcl/tk, python/jython, to increase productivity.**
- **Understand what's important to customer - quality, cost, response time, etc. - and apply data analysis techniques to identify areas for improvement.**
- **Introduce discipline so that defects are prevented or if introduced will be detected and removed quickly. Use on-going tested and working software to monitor real progress in the project. Do continuous improvement.**
- **Understand modeling and statistical process control and apply them to those defect removal processes, such as testing and inspection, where these techniques are applicable.**

- **Build trusting work relation, negotiate win-win solutions, empower team and develop people for happy workers and highly effective teams.**

Chapter 8

Lessons From Stories

The following are a few stories where useful lessons are extracted from full-length books. The original source are cited in reference at the end of each story. Hopefully, the short sketch below would whet reader's appetite to want to read the original.

8.1 Don't Go Nuclear - Lessons From the Cuban Missile Crisis

Most people don't realize that during the Cuban Missile Crisis we were really just a hair's breadth away from total nuclear war! The history of that episode of Cold War contains great lessons about how to de-escalate during a crisis.

Here's some background on what led to the crisis in the first place. It was triggered by the installation of 15 Jupiter Intermediate Range Ballistic Missiles (IRBM) in Izmir, Turkey. This was

intended to strengthen the relation between US and her ally Turkey but was considered a personal affront by Soviet's Premier Khrushchev. While Soviet assured US that they had no plan to install missiles in Cuba, secretly, shipment and build-up started shortly after the Turkey installation. In hindsight, these installation in Turkey may be ill-conceived as the technology was dated and the same protection and coverage could have been provided by US nuclear submarine. In fact, US agreed to withdraw these missiles as part of a secret deal later on.

On Oct. 16, 1962, US reconnaissance plane found solid evidence of Soviet nuclear missile installation being constructed in Cuba. Those were judged to be mid-range (1500 miles), offensive, but not yet operational. On Oct. 24, President Kennedy announced the blockade on Cuba and surrounding area by US Navy to prevent further shipment from Soviet Union. The word "quarantine" was used in the actual announcement as "blockade" is a word considered to be a form of declaration of war, and President Kennedy did not want to do that. On Oct. 25th US also presented evidence of the offensive missile installation at an emergency session of the UN Security Council. However, Soviet ships continued to move toward Cuba, and the crisis was coming to a showdown.

On Oct. 26th, President Kennedy received a private letter from Premier Khrushchev to withdraw the missiles in exchange with US guarantee not to invade Cuba or support such invasion. However, before President Kennedy had replied, a second offer was announced on public broad-

cast the next day that included both the proposal above plus the condition of US withdrawal of missiles in Turkey. While all this was happening, an U-2 plane from US was shot down over Cuba by Soviet missile, and the pilot Major Rudolf Anderson was killed. Some in US inner circle of power called for immediate invasion of Cuba to revenge this incident.

During that time, the CIA did not think there were nuclear warheads ready in Cuba. US was poised for massive bombing and invasion on Oct. 29th. Just hours before the strike time, Premier Khrushchev announced that installation will be dismantled and Soviet ships started to turn back. A short time later, the blockade was lifted. A crisis was avoided. Yet most did not know that the world was just a hair's breadth away from mutual total destruction!

Before we jumped to the conclusion that Premier Khrushchev backed down under pressure, let us first look at some remarkable revelation about what really happened during the crisis. This came about through a remarkable program at Brown Univ., called Oral History Project (OHP) (choices.edu). In order to learn historical lessons from the people involved with the event, the OHP program has sponsored a number of conferences to bring key players from major conflicts back together. There was a 1992 conference on Cuban Missile Crisis. From this and other conferences, we learned that not only was there 162 nuclear warheads ready, with 90 tactical warheads, at the time in Cuba, but Fidel Castro would insist to use them if US attacked Cuba, knowing full well that the result would be total destruction for Cuba.

We also learned that instead of the estimated less than ten thousands Soviet troops, there were actually close to fifty thousand in Cuba. The planned US attack did not include tactical nuclear warheads. Imagine what would happen when US attacked and Cuba and Soviet Union responded with tactical nuclear warheads?! US would be compelled to "go nuclear" also. Then Soviet is likely to respond with tactical warheads on US missile installation in Turkey and other places, and NATO will respond in kind, and things would escalate and unravel from there. From these conferences and lessons from history, we can draw our first lesson in crisis management - "YOUR ASSUMPTIONS AND INFORMATION ARE OFTEN WRONG!" Subsequently, hotline was installed between White House and Kremlin in order to have a direct channel of communication between world leaders and to avoid accidental attacks due to misunderstanding.

How was the crisis resolved? Here is where sound understanding about your opponent is so crucial. In his cabinet meeting, President Kennedy was quite concerned about Khrushchev's second offer. He reasoned that with the second offer, Khrushchev won't take out the Cuban missiles with just the no invasion guarantee, and confrontation between the two superpowers may be unavoidable. However, Tommy Thompson, an old hand in US diplomacy with Soviet Union and former Ambassador to Soviet Union, thought otherwise. He argued that Khrushchev could tell his people, "I saved Cuba, I stopped an invasion.", and that's enough ground for him to back down. Even though Tommy

was lower ranking and not even a cabinet minister, President Kennedy was able to recognize Tommy's special expertise about the inner working of the Soviets and listened to his advice. President Kennedy decided to respond to the first offer in public, but also sent Robert Kennedy to tell Soviet Ambassador in US in person that the Turkey missiles will be dismantled as a separate and private deal. So another important lesson on crisis management is "HAVE EMPATHY, TRY TO UNDERSTAND WHAT PROBLEMS YOUR OPPONENTS ARE FACING". Just rely on the rational analysis alone is not enough.

There's yet another important lesson to be learned from the Cuban Missile Crisis, which is "THINGS COULD EASILY GET OUT OF CONTROL. TRY NOT TO PROVOKE. DE-ESCALATE. DON'T GO NUCLEAR" In shaping a US response, President Kennedy assembled cabinate members and other key officers and asked them, if possible, to come up with a single response. The team could not come to agreement and presented Kennedy with two options. One was immediate massive invasion, and the other was the blockade. President Kennedy chose the blockade route as he did not feel it's justified to come to blows over some outdated and unnecessary missiles in Turkey. Imagine what would have happened if a different leader chose to follow the immediate massive invasion proposal.

During a major crisis, there are many opportunities for accidents to happen, and things could easily get out of control. During the Cuban Missile Crisis, there were at least three incidents that could escalate into major problems, but fortu-

nately did not. First incident was the shot down of U-2 plane over Cuba by Soviet missile, causing the death of the pilot, Major Rudolf Anderson. There were many in the US crisis management team that would like to "reply" to this incident by a full scale invasion. However, cool reasoning of "if we do this, and they do that, then what's next?" prevailed, and no action was taken to respond or escalate. It's necessary during crisis management to think through one's move like a chess master. Because in real life, just like in chess, one thing would lead to another. "ONE MUST THINK THROUGH ONE'S MOVES", as things could easily get out of control.

There's also a second incident. A US spy plane strayed into Soviet Union and was almost intercepted, even though President Kennedy issued an moratorium on flying such planes into Soviet Union in order to avoid escalation.

The third incident during the crisis was the hunting of a Soviet submarine by US destroyers near Cuba. Depth charges were dropped in order to force the submarine to surface. Unbeknown to US, this submarine was equipped with nuclear-tipped torpedo. The ship was authorized to fire if all three top officers were in agreement to its use. Fortunately, in a story similar to the movie "crimson tide" - Or perhaps the movie was inspired by this incident - one of the officer was against the use of the torpedo, the other two wanted to fire, and a war incident was avoided. We were again just a hair's breadth away from nuclear war!

Aside from the lessons on crisis management, the Cuban Missile Crisis chillingly exposed the fact that how close and how easy we were to to-

tal mutual annihilation. With powerful weapons such as tactical nuclear war heads so numerous and so wide spread, the world remains an extremely dangerous place. Robert McNamara, a key player as US Defense Minister during the Cuban Missile Crisis, with James Blight, the professor behind the Brown Oral History Project, in their recent book, "Wilson's Ghost", argued convincingly that nuclear weapons, strategic or tactical, no longer have a role in the world today, and should all be abolished. They advocate multi-lateral consultation for collective action on security issues as history showed again and again, "ONE-SIDED UNI-LATERAL ACTION OFTEN LEAD TO UNITEDNED, TERRIBLE, RESULTS".

References - In addition to "Wilson's Ghost", there's also a good DVD "The Fog of War: Eleven Lessons from the Life of Robert S. McNamara", directed by Errol Morris (2003). A good web site for additional information and links is /en.wikipedia.org/wiki/Cuban_Missile_Crisis. Web site for OHP is choices.edu.

8.2 The Start of First World War - A Cautionary Tale of Unintended Consequence

Hours before First World War to break out by Germany's attack of France, German's Emperor Wilhelm Kaiser got cold feet to fight a two-front war with France and Russia at the same time. He asked his generals to halt the invasion on the West

front with France. His generals told him, "it's too late to stop now!." In fact, His Chief of General Staff, Helmath von Moltke, was so upset by the request, he told others that he'll throw away his phone, so the Emperor could no longer reach him.

As a result of the First World War (WWI), ten millions were dead, three great empires were crumbled (Germany, Russia, Austria-Hungary), and the political face of the world completely changed. None of the key players had wanted or anticipated such outcome, yet since the first incident, the world seemed to march rigidly and inescapably toward colossal disaster, with the players as powerless and helpless like the victims in a Greek tragedy. To understand why that's the case, and to learn lessons from that, let's first review briefly the key events that led to WWI.

On June 28, 1914, Franz Ferdinand, Crown Prince of Austria-Hungary, and his wife Sophie, on Ferdinand's official visit to Sarajevo, a city in the empire not far from Serbia, were assassinated by Gavrilo Princip, a member of Serb nationalist of secret society Black Hand. Long troubled by the harboring of Serb nationalists in the neighboring Serbia, and after securing an iron-clad guarantee of support from Germany, Austria-Hungary delivered an harsh ultimatum on July 23rd to Serbia, to be replied in 48 hours. On July 25th, Austria-Hungary not satisfied with the answer given by Serbia, broke off diplomatic relationship with Serbia. On July 28th, Austria-Hungary partially mobilized and declared war on Serbia. In the mean time, Serbia Prince Regent Peter interpreted this as Austria-Hungary's attempt to annex Serbia, and appealed to Russian

Czar Nicholas II, who was also Willy Kaiser's cousin, for help. Russia, bound by treaty to Serbia, then partially mobilized. Germany, bound by treaty with Austria-Hungary, declared war on Russia on August 3. More countries were drawn into the conflict as France had treaty with Russia and England had treaty with France. The war was escalated into a global conflict even though none of the parties intended to do that in the beginning.

The treaty system between nations certainly locked nations into obligations, and that was an important factor in the continuous escalation of conflicts. The only country that got out of it initially was Italy, which had treaty with both Austria-Hungary and Germany, but only if they were attacked. Since in this case, both were the attackers, Italy used the clause to get out of it. In fact, Italy joined the other side a little later. But why the nations came to blow in the first place? And why should Germany's Kaiser gave Austria-Hungary such iron-clad guarantee?

There were several factors that influenced Germany and Austria-Hungray's harsh stance toward Serbia. Austria-Hungary's Emperor Francis Joseph, aging, war weary, and sick, although urged by his ministers, originally was reluctant to mobilize or to take action against Serbia for fear of involving Russia. Action was delayed till Germany's iron-clad guarantee. This gave Austria-Hungary confidence to proceed. But why should Kaiser provided such strong gurantee? First, Ferdinand was a personal friend of Kaiser, who liked the couple a lot and Kaiser had just visited the couple shortly before the assassination. So his

personal grief and anger was a big factor. But most importantly, in Kaiser's mind, he could not imagine Russia's Nicholas, his cousin, could be sympathetic to any act of violence against the royalty. In fact, he did not even bother to check with Russia's intention nor imposed any restraint on Austria-Hungary's action. He was so confident that the conflict would be local and resolved quickly that he left for vacation.

Kaiser gave Austria-Hungary an "guarantee by blood and honor". This was essentially a blank check for Austria-Hungary, and he urged them to act quickly toward Serbia. Austria-Hungary, the weaker party in the partnership, was very much interested to teach Serbia a harsh lesson, as a way to regain some of her former glory as an empire. So the term of the ultimatum was extremely harsh and with terms like free search in Serbia that violated Serbia's sovereignty and Serbia could not possibly accept.

Both Austria-Hungary and Germany had envisioned this as a local conflict, involving only Austria-Hungary and Serbia. However, they forgot that Russia was bound by treaty with Serbia, and both are countries with major Slav ethnic component. Furthermore, with the recent defeat of Russia's Navy by Japan (1904 - 1905) near Manchuria and Korea, Russia was very much in need to prove her military might. This reminds us the lessons, "YOUR ASSUMPTIONS AND INFORMATIONS ARE OFTEN WRONG!", "HAVE EMPATHY, TRY TO UNDERSTAND WHAT PROBLEMS YOUR OPPONENTS ARE FACING". Just rely on rational analysis is not enough. It was a most serious

failure on Kaiser's part in judgment and communication!

But why the war was fought in so rigid a manner? The rigidity in the way wars were fought has to do with the technology and military thinking of the period. It's pretty much a war of infantry and land based defense (trenches). Each country planned ahead for the next war to the last details of all the logistics. In Germany's case, a two-front war with both France and Russia had been anticipated in the Schlieffen Plan, which was the guiding light for Germany's planning. The strategy there was to attack Frances first, decisively and quickly, and to win in the West front in about five weeks, before Russia has completed the mobilization of its huge army, which would take six weeks. The plan there had always been to move all the troops and equipments by train to the West front first, with all the logistics of movement in trains planned to the last details. That was why when Kaiser went sour with Russia and would very much like to attack Russia only, to avoid drawing in Frances and England, his generals told him there was no way they could reverse course and move the troops to the East front instead. There's no such plan! Besides, due to the treaty systems, they expected Frances and England to be involved sooner or later. The Plan called for to defeat France first so Germany did not need to fight on two fronts at the same time. In fact, in order to reach France quickly, on August 4, 1914, Germany violated Belgium's neutrality. Because of this invasion and an old treaty between England and Belgium, England declared war with Germany that day, exactly the outcome

Kaiser very much liked to avoid. This reminds us the lesson, "ONE MUST THINK THROUGH ONE'S MOVES", "THINGS COULD EASILY GET OUT OF CONTROL."

There was another mistake that prevented a diplomatic solution to the conflict. And that was Germany's belief of a First Strike advantage. If one has to fight, then the thinking goes, the one who strikes first and decisively, will win. The problem is, with that approach, not only things are likely to escalate because of the first strike provocation, but other solutions by diplomatic means, which would take time, have no chance to work. This is the case with Germany's Kaiser. While he attempted to defuse the crisis by exchanging messages with Russia's Nicholas, he did not give it enough time for the peaceful gestures to work. He delivered a twelve hours ultimatum on July 31st for Russia to roll back it's partial mobilization. At the end of that, with no positive response from Russia, he decided to strike first, declared full mobilization on July 31st, and escalated up the conflict to another notch. This reminds us the lesson, "ONE-SIDED UNILATERAL ACTION OFTEN LEAD TO UNITEDNED, TERRIBLE, RESULTS", "TRY NOT TO PROVOKE. DE-ESCALATE. DON'T GO NUCLEAR". Once the war started, the slaughter begins, the course of the war was out of anyone's control, and the results were totally disastrous and unpredictable.

References - John Stoessinger, "Why Nations Go To War", St. Martin's Press (1974). Also, see articles at http://www.firstworldwar.com/origins/.

8.3 Grameen Bank - Lending Money A Little Differently

John Chambers, CEO of Cisco, characterized innovation this way in his talk at MIT in 2005, "INNOVATION IS NOT YOU'RE SMARTER OR WORK HARDER, BUT TO UNDERSTAND HOW OTHERS FAILED IN THE PAST, AND DO THINGS A LITTLE DIFFERENTLY."

An excellent example of this concept is the approach of Grameen Bank, a bank devoted exclusively to provide financing to the poor, especially women. By conventional wisdom, the poor, having no collaterals, are considered high-risk for bank loans. Yet Grameen Bank found a way to make bank loan viable, with an astonishing low default rate (less than 2%). Considering the fact that this work was started in a poor village Jobra in Bangladesh, where there's strong tradition that women have little rights in society, and in some areas, they are not even allowed to talk to strangers face to face, the success was especially astounding, and almost like a miracle happening. Yet the success is no accident. The Grameen Bank approach has been successfully replicated world wide, country after country, and the microcredit concept Grameen Bank has pioneered is now a world-wide movement, as well as a center piece of United Nations and many country's effort and policy to help to lift the poor from poverty. Grameen Bank asks for no collateral, borrowers sign no paper, and Grameen Bank even loans money to beggars! So why is Grameen Bank

successful where conventional banking failed?

To understand that, we need to first go back to see how Grameen Bank got started. Grameen Bank was the creation of Muhammad Yunus, a professor of economics at Chittagong University near Jobra. In response to the dire cycle of poverty poor villagers were trapped in, his action eventually resulted in creation of the Grameen Bank, or Village Bank in Bangladesh. Prof. Yunus got his advanced degrees in US and returned to Bangladesh in 1972 to teach. In 1974, the country fell into a wide spread famine. Prof. Yunus recalled the frustrations he had that he was teaching all these fancy economic models in a nice classroom, yet it had so little to do with the difficult lives of poor villagers right next to the campus. In his own words, "I felt the emptiness of those theories in the face of crushing hunger and poverty. I wanted to do something immediate to help people around me, even if it was just one human being, to get through another day with a little more ease." By 1976, Prof. Yunus regularly visited Jorba with his students in order to better understand what made their lives so difficult and how they could help. For example, he was shocked to learn that a poor woman, Sufiya Bergum, was trapped in poverty for the lack of five taka, the equivalent of twenty-two US cents. He said, "I had never heard of anyone suffering for the lack of twenty-two cents. It seemed impossible to me, preposterous." Sufiya made beautiful bamboo stools for a living. But since she was poor and could not borrow from the bank, she needed to borrow from the money lender. Each day she borrowed twenty-two cents to buy the materials

for her trade. However, She was obligated to sell the products she made back to the money lender, and earned only two cents, barely enough to feed her and her children. So in fact, she was like a bonded slave. In contrast to the common belief, Prof. Yunus found out that *it's not that the poor don't want to work or lack skills,* but for lack of source of capital, that they are trapped in a perpetual cycle of poverty and are essentially no different from slaves in bondage. They cannot start to improve their lives by earning the full values of their labor in the free market because of the lack of source of credits. So he asked one of his student to find out all the people in the same situation as Sufiya in Jobra. There were forty-two people with a total need of only twenty-seven US dollars! He gave them interest-free loan to help them to break this vicious cycle of misery. Later on, when this and other small pilots were all working well, and villagers were able to repay the loan, he wanted to make an institutional solution, such as a bank, in order to solve this problem in vast regions of Bangladesh. That was where he ran into the most resistance and skepticism of his endeavor.

The bankers he talked to simply don't believe this approach is viable. Like many others, I have applied mortgage for a house several times before. It's a complicated process with lots of paper work. Bankers are very careful with their money. They want to find out about what you make each month and what assets you have and your credit history in order to decide if you could afford to pay back the loan and if you are creditworthy. Lawyers are involved and one has to sign many legal documents of obligations with a lot

of penalty clauses. So bankers by nature are a very conservative and cautionary bunch. They think Yunus is crazy to lend money to poor people. They told Yunus he would lose money big way since poor people has no collaterals nor skills to earn money to pay back the loan. They also pointed out that the banking cost would be too high for such a small loan. They told him Jobra is different. The scheme might work in Jobra, next to the University with free college student volunteers, but not elsewhere. They told him the poor, especially women in Bangladesh, are mostly illiterate and have never handled money before, so how do you do banking with them? They can't read or sign any papers. Where do you find workers willing to go to village to lend loans, as one can't expect the poor women, who seldom leave their houses, to know how to come to the bank to apply for a loan. They politely advised Prof. Yunus to focus on economic theory and leave the banking business to them, the experts. In the face of such criticism and resistance, most people would have given up. But Prof. Yunus was no ordinary people. He understood why conventional banking procedure won't work for the poor villagers but he could "LEND MONEY TO THE POOR A LITTLE DIFFERENTLY".

For a while, he personally guaranteed all the loans the bank gave to the poor villagers and signed all the papers for them, But eventually this led to the formation of Grameen Bank in 1983. As of April, 2006, Grameen Bank has 6.04 million borrowers, 96 percent of whom are women. With 2014 branches, Grameen Bank provides services in 65,847 villages, covering more than 97

percent of the total villages in Bangladesh. Borrowers of Grameen Bank at present own 94 per cent of the total equity of the bank. The remaining 6 percent is owned by the government. Since 1995, Grameen Bank no longer accept any donor money and all loans were financed from deposits. Projected disbursement for 2006 is $821 millions in US dollars.

There are many key differences between Grameen Bank's approach and conventional banking. Instead of collateral and legal instruments, Grameen Bank asks borrowers to form five-members group to support each other, but there is no joint liability by the group for each individual's loan. In case of difficulty to meet loan payment, instead going into legal action, Grameen Bank workers help borrowers to reschedule the loan and get over the difficulties. Grameen Bank's goal is not to maximize profit but to make financial services to the poorest, especially women, and has many other products and services to improve the welfare for the whole family, like health, education, insurance, pension. Grameen Bank has its branches located in rural villages, and workers go to the villages to meet the borrowers instead of the other way around. Paperwork to keep track who has how much money deposited or borrowed were greatly simplified as many villagers are illiterate. Various innovation, including IT, were introduced to reduce the book keeping work load for the workers. The overall success of the project, I think, showed that the following Grameen Bank's premise is indeed correct, namely, *each person, no matter how poor, has endless potential, and will not abuse the help*

and opportunity to lift oneself out of poverty.

It would be naive to think that once one find the right magic idea, the rest is a piece of cake. To bring about the success of Grameen Bank and micro-credit, Yunus faced many more challenges beside institutional resistance. Many women literally refused to borrow money from the bank. They want to defer to their husbands. There were many natural disasters that made loan repayment impossible. There were cultural barriers for men to talk to women, for women bank workers to walk alone in village, or to continue working after marriage. Yunus and leaders of Grameen Bank need to learn and innovate continuously as they are doing path-breaking work that no one else has been there before. The lessons they learned form the basis for the replication program for other regions and countries. The key innovation is the trust placed in poor people and the mechanism to make repayment easy (frequent repayment at very small amount, almost no paper work, bank clerks go to the villagers). Yunus has a program to demonstrate that even beggars could be helped to use loan to become business person, with equally low default rate.

However, in spite of the hardship in working condition, (there's no "banker's hours"), recruiting Bank workers was never a real problem. As pointed out in "The Art of Business", Grameen Bank is not only doing things right but is also doing the right things. They are changing people's life for the better with their "Sixteen Decisions" for Grameen Bank members, such as "We shall plan to keep our families small. We shall educate our children, etc." So Grameen Bank has no dif-

ficulty to recruit young and energetic people. It's not just an unusual banking business, it's a life uplifting adventure.

The success of Grameen Bank and micro-credit movement showed clearly that there's a huge need for credits by the poor which was not met by conventional banking or government and society at large. Furthermore, the success also showed that there's a tremendous store of good will in people that could be mobilized to help address the needs of the poor. Certainly, in the early phase of Grameen Bank, employees worked more like dedicated volunteers, but Grameen Bank never had any real difficulties in attracting qualified people to work at the bank. The Grameen Bank story has a very positive message for us all - namely, poverty is a solvable problem, poor people can be trusted with credit, and once helped, can get out of poverty and make useful contribution to society like everyone else. One just need to find a way to let the poor to help themselves. Micro-credit is certainly one very important way.

It's great that this work of Prof. Yunus and Grameen Bank has now been recognized worldwide by their being awarded the Nobel Peace Prize in 2006. Lessons to take home with, "CONVENTIONAL THINKING MAY BE WRONG, THERE MIGHT BE A BETTER SOLUTION IF WE DO THINGS A LITTLE DIFFERENTLY."

References - Muhammad Yunus, "Banker to the Poor", Perseus Books Group (1999), also articles from http://www.grameen-info.org/. Also, talk by Muhammad Yunus, "Ending Global Poverty" at MIT,

http://mitworld.mit.edu/video/289/ and 2006 Nobel Prize lecture at nobelprize.org. Raymond Yeh and Stephanie Yeh, "The Art of Business - In The Footsteps of Giants", Zero Time Publishing, 2004. John Chambers' talk - "The Power of the Network to Change the Way We Work, Live, Play, and Learn", at http://mitworld.mit.edu/video/293/.

8.4 Ashoka - To Empower Thousands of Social Change Makers

Social entrepreneur is a new ideal for many young people. Instead of making a lot of money as a life goal, many are trying to apply the same innovative, hard driving, entrepreuring approach in forming new business to form new ventures for social changes to improve people's lives. To understand the social entrepreneur phenomena, there is no better place to start than to visit ashoka.org or to read David Bornstein's book about Ashoka, "How to Change the World". For Ashoka, founded by Bill Drayton in 1980, for "Innovators for the Public", has the longest history in doing this kind of work, and has the largest network world-wide of social entrepreneurs. Since its formation, the organization has identified and supported over 1,800 Ashoka Fellows in over 60 countries.

Bill Drayton demonstrated leadership at an earlier age. He launched "The Sentinel", a class newspaper, in grade four. The newspaper

soon grew from two-page to thirty-two-page with whole team of classmates as writers, illustrators, and with advertisement from local merchants, and even got it distributed to some other schools. Public work and history about India has always been of particular interest to Drayton. Another defining experience was a trip to India in 1963, when he was 20, to follow Vinoba Bhave, a key disciple of Gandhi, to walk from village to village. Bhave was applying nonviolence approach in land reform. Through his effort on land gift and village gift movement, by 1960, seven million acres of land were redistributed voluntarily to support landless people and "untouchables". Over the years, Drayton came to believe that Gandhi had this great insight that our age calls for ethics based on empathy instead of relying only on rules, and empathy could be a very powerful force to change society for the better. After graduation from Harvard and post-graduate studies at Oxford and Yale, he worked as McKinsey management consultant on public issues during the early 1970s. In 1984, Drayton was awarded the famous five-year MacArthur "genius" Fellowship for his work as public service innovator.

Drayton has been a social change maker himself. Drayton was always interested in the political process and had worked on several campaigns. In 1977, he was appointed as Assistant Administrator of Environment Protection Agency (EPA). During his two years there, his ability to look at a problem and solve it in a fundamental way was demonstrated by having his idea on "bubble" - to allow trading in pollution control - enacted into US environmental policy. The concept of "bub-

ble" is to create incentives for polluting business to control pollution by lumping the burden of pollution from all processes of the business for a given pollutant (say Greenhouse gas) together and allow the business to find the cheapest way to meet the set target, such as to fix those processes that are least costly to clean up first. This innovative idea was then hotly contested by environmentalists, EPA personnel and many others. Only through his hard work, political skill, and tenacity, that the approach was adopted as policy. Today of course emission-trading is a central feature of the Kyoto Protocol. The emission trading policy in the 1990 Clean Air Act had brought significant reduction in sulfur dioxide pollution.

Drayton was also tenacious to fight for the integrity of EPA as an organization. After Reagan became President in January 1981, it quickly became clear that the Reagan Administration was planning to destroy EPA by drastically reducing its budget. Drayton understood what was going on and rose up to form Save EPA to fight this. He explained, "They couldn't win the policy fight, so they were going to destroy the institution." "I like to build things. But I had spent a good part of my professional life building the environmental institution at the municipal, state, and federal levels. And what they are doing was illegitimate; it was just *wrong.*" Following advice from a friend, that the key in the fight is "to make it obvious to them that this is going to be political torture until they stop", Drayton, over the next three years, mobilized media to keep close watch over EPA budget and keep the heat up about the danger of destruction of EPA. This topic even got

into *Doonesbury* comic strip. The EPA had lost a third of its funding. Drayton said, "They did tremendous damage, but it could have been a lot worse."

For Drayton, it's a compelling idea to apply the concept of venture capital firm to fund social entrepreneur work. Given his background and track record, he was just the right person to pioneer this new field of social change making. In venture capital, one seeks high yields from modest but focused investments by leveraging other's great business ideas. In funding social entrepreneur, "ONE LEVERAGES OTHER'S GREAT SOCIAL CHANGE IDEAS", and the return is not measured in money, but in long-lasting and wide-spread social change. But the power of leverage is the same. Apply a small amount of resources over a few years, to the right people with the innovative idea, commitment, and moral fiber, at an very early stage of the venture, so they could devote full time to bring their ideas into fruition to achieve large scale and long lasting impact. Furthermore, by doing this over long period of time, and by forming strategic partnership and networking with business and citizen sector organizations, there are *further leverage at group and sector infrastructure level.* The global network of Ashoka Fellows are now a tremendous resources to help Fellows to solve problems in their work. Ashoka's partnership, such as with McKinsey, also provide vital input and support to nurture the new social entrepreneur organizations in its formative years. So Ashoka provides leverage on many fronts - in venture capital, in seed money support, in social enterprise incubation, in lead-

ership skill training (Ashoka's Global Academy), and in global networking.

Drayton chose the name Ashoka for a good reason. Ashoka was the name of a third century B.C. Indian emperor, who set an example for global thinking, tolerance, and innovation in both economic development and social welfare. In Sanskrit, Ashoka means the "active absence of sorrow". Emperor Ashoka was a person who knew the how-to to get things done. He played a seminal role in the spread of Buddhism. Although he himself was a Buddhist, he guaranteed freedom of religion in his empire. He established the world's first large-scale class of civil servants devoted to public welfare. They built India's Grand Trunk Road, from Afghanistan to West Bengal, and provided support such as water, shade trees, and rest houses, along much of the length of the road. They also built hospitals, and did land settlement work. Drayton also chose oak tree as the organization's logo, to symbolize "from little acorns do great trees grow".

While the idea of Ashoka came naturally to Drayton, to get it funded or to find qualified fellows was very difficult in the beginning. Drayton started Ashoka with $50,000 of his own money and some private donation. For the first five years, he could not get a single public foundation to support it. Today in 2006, it has a budget close to $30 millions US dollars. To recruit people in other countries to participate was difficult. There were a lot of suspicion on whether Ashoka might be a cover of CIA or some other covert work of USA. Since Ashoka are breaking new ground in the social change making field, many new things

have to be created, such as how to find, select, and review candidates for the Ashoka Fellows? How to support them and for how long? New systems and support infrastructure need to be invented.

Let's now look at the current process of Ashoka Fellow selection. According to ashoka.org web site, Ashoka Fellows are funded at the launch stage of the social enterprise, typically to provide a living stipend for the Fellows for an average of three years to allow the Fellows to focus full-time on building their institutions and spreading their ideas. In addition, Ashoka also provides the Fellows a global support network of their peers and professional consultants, and once elected, Fellows are part of the Ashoka global network of Fellows for life. Ashoka used the following five criteria to evaluate potential candidates for Fellowship:

- The Knockout Test - Look for innovative idea or solution to social problems that could change the field.
- Creativity - Does the person have a track record of compelling vision and creative in problem solving?
- Entrepreneurial Quality - Are the leaders totally passionate and dedicated to realize their social vision?
- Social Impact of the Idea - The change idea must have potential of national or broad regional impact.
- Ethical Fiber - The Fellows selected must be totally trustworthy.

In addition, Ashoka will not support anyone who is violent, or a partisan political leadership, or support violence, discrimination or totalitarianism. To find potential candidates, Ashoka has built up over the years an extensive global nominator network, consisted of partner organizations, business, social entrepreneurs, and community leaders.

How successful are the selection process and what impact Ashoka had? Each year, Ashoka routinely survey and interview Fellows selected five years ago and ten years ago to conduct Measuring Effectiveness Study. The following are the composite results collected over last six years (1999 - 2004). The results (all for ten-years post-selection) are very impressive indeed:

- The Original Vision - 83% Fellows are still working at the original vision after ten years.
- Independent Replication - 82% of Fellow's work have been independently replicated.
- Policy Influence - 71% of Fellow's work are adopted as government policy.
- Leadership Building - 66% Fellows are now leaders in their field.
- Ashoka Leverage - 77% considered Ashoka's overall support critical or significant to their work.

The success of Ashoka and its Fellows is a tremendous reminder for us not to despair in today's world that is full of conflict, violence and trauma. It's easy to lose heart reading the

daily reporting of wanton slaughter or violence in the news. However, there are thousands and thousands of social entrepreneurs working tirelessly and ceaselessly to improve the lives for millions. The world has the capacity and ability to make it a good place to live for all. As Drayton pointed out, we must use *empathy* as the new guiding ethical principle for the 21st century. Ashoka could make this tremendous accomplishments only through the principle of leverage. No matter how smart or capable the individual is, he or she could only personally do the social change work of at most a few of the Ashoka Fellows. But by leveraging at multiple levels, the organization now has world-wide impact and is a major force in the new field of social change making. Lessons to take home - "APPLY THE PRINCIPLE OF LEVERAGE: INVEST IN A NUMBER OF SELECTED PROJECTS, RELATIONS, OR WORK THAT POTENTIALLY COULD LEAD TO HUGE BENEFITS IN THE FUTURE".

References - David Bornstein, "How to Change the World - Social Entrepreneurs and the Power of New Ideas", Oxford Univ. Press, 2004. See also the web site ashoka.org.

8.5 Greg Smith - How To Survive Catastrophe And Live To Tell

Most people don't function well facing catastrophe. They are overwhelmed, confused, paralyzed. They feel shock, despair, anger, but es-

pecially powerless. Because of that, frequently, they do nothing and just let the catastrophe to take its natural course of destruction. But there could be another way. The chance of survival are better if we are prepared. One way to prepare is to learn from the stories of people who survived catastrophes and lived to tell their stories. The story below is part of the extraordinary life of Greg Smith.

In Dec. 1986, Greg Smith, who was 34 at the time, was told that he had an inoperable brain tumor and had only three months to live. Apparently, his benign brain tumor, which had been there for more than a decade, had suddenly turned malignant and went on exponential growth unexpected and undetected. Now, he was told, it's too late to operate. Furthermore, he learned all this from the doctors at the Mayo Clinic, a top medical institution in the country. Yet he lived and wrote his book "Making Miracles Happen" in 1997, to share his experience of survival to help others. He also lived to see the book he was working on back then during the crisis, "Jackson Pollock: An American Saga", to get published and he received Pulitzer Prize for the book in 1991.

Now there's something about Greg Smith that made him an excellent teacher for others to fight catastrophe. He was young, loved life and desired to live. He also had a special talent and tenacity to dig out information. When he was researching the book on Jackson Pollock, he and his co-author and partner Steve pledged that, "we would go anywhere, talk to anyone, read anything, follow any lead, turn any stone in the search for options." That tenacity will be crucial when one is

tested by life with catastrophe. Both of them are lawyers. Back in 1983, they have already applied that same gift and persistence to write a book, called, "The Best Lawyers in America". While there may not be a miracle for everyone with an inoperable brain tumor, one does need some grit and character to make a miracle possible. The story of "Chasing Daylight" by Eugene O'Kelly is equally moving, but the outcome was very different. However, in spite of these caveat, the way Greg Smith went about to create his miracle is very instructive, and is the story given below.

In facing a tragedy or catastrophe of such magnitude, it's natural for people to give up, "to pack up life and get ready to die". However, whether one is naturally a fighter or not, the first lesson from Smith's story is to "TAKE BACK CONTROL", for "LEARNED HELPLESSNESS KILLS!". Many died needlessly because they have given up. Smith cited experiments done by Madelon Baranoski at Yale which showed that rats subjected to random shock they have no control would die at high rate (75%), but the death rate would be much lower (25%) if the rats had some mechanism of control. Similar results were implicated in people too. Stress level became very high when people lose autonomy or control of their lives.

What kind of control could one find when you were told by world-renown authorities that you have only three months to live? It turns out that, in almost any dire situation, there's always something one can do. Just the process of looking for options, second opinions, and assessing and analyzing alternatives by itself is very helpful. The

mindset would be very different if one has evaluated all the facts and options and then choose to not to go through "heroic" rescue effort, because then the choice is made by oneself, not by fickle fate. In Smith's case, or in any medical situations, Smith pointed out that there are usually a lot of options available.

First, there is the choice of doctors. Each doctor, even for the same specialty, is different. Not only the training, skill, and experiences are variable, but also the supporting environment of the clinic, supporting staff, or the hospitals. Secondly, there's almost always different views on how to treat a problem or the assessment of outlook. For life-threatening illness, it's critical to get several opinions. Third, medical science is always moving into new experiments and discoveries. There might be a lot of experimental procedures and drugs not yet available to the general public, but available through various trial programs. For all these reasons, one must do one's homework to "RESEARCH AND DEVELOP REAL OPTIONS" so one's decision is based on thorough knowledge and not to be dictated by the situation or the first doctor. Only by taking back control this way, whatever happens, one would know that one has done the best one could at the time. In medical science as in many other life situations, there is no 100% certainty, and what a lot of doctors are saying are just their best guessmates and not a sure thing. There are always things one can do to increase the odds for one's survival and success.

To find real options one does need to take some "PERSISTANCE AND PERSERVER-

ENCE" (P&P), but then, after all, it's your own life you are fighting to protect! To reach the various doctors for Smith to develop his own options for treatment, he just forced himself through phone to every doctors he could find that know anything about brain tumor and how to fight them. And he didn't take "No" as an answer. He remembered that once he insisted on to talk to a neurosurgeon directly, and he told the secretary "it's a life-or-death situation", the secretary replied tartly, "I know, I know. They are all dying". It's through such effort that Smith found out about the experimental procedure of Doctor Sadek Hilal at Columbia-Presbyterian who has a new procedure to inject special silicone into the blood vessels going into the tumors to starve them, a procedure called therapeutic embolization. After meeting Dr. Hilal and evaluating all the options, Smith chose to do the procedure in March 1987 and got his stay of execution. Even through there were still other complications down the road and the tumor needed to be "maintained", he got his life back, and each year he lived after that would be one more year he might not have otherwise. After learning a story like that, shouldn't we all be thankful each day we are healthy and able to do the things we like to do? Critical illness and facing death has so much to teach to us about how to live our lives.

In Smith's time, a lot of the data was not there or easily accessible. In today's world, data are more accessible through the world wide web and public resources such as the National Institute of Health. Also, medical professionals are now more comfortable with patient's need for

information, second opinions, and taking control back. The patient's recovery is now more like a partnership between the doctor and the patient rather than an one-person show of the doctor's brilliance and heroic effort in rescue. As for inforamtion sources, there are "The Best Doctors in America" by Greg Smith and Steven Naifeh. There's the Physician's Data Query (PDQ) on all experimental programs for cancer treatment in US from National Cancer Institute (NCI) (http://www.cancer.gov/cancer_information/pdq/), and there's the "Outcome Data Bank" at National Institute of Health (NIH) about cure rate, survival rate on various procedures. For other catastrophes, other data source would be necessary, but the principle of *taking control back and developing real options* are the same.

References - Gregory White Smith, Steven Naifeh, "Making Miracles Happen", Little, Brown and Company (1997).

References and Notes

[ADA] Jonathan Adams, Srinivas Koushik, Guru Vasudeva, George Galambos, "Patterns for e-Business - A Strategy for Reuse", IBM Press, 2001.

[ADE] Gillian Adens, "The Role of Risk in a Modern Software Development Process", /downloads/TheRoleofRisk.pdf page at www.tassc-solutions.com.

[AGI] Agile Alliance home page at http://www.agilealliance.org/home. For agile process, see /resources/articles/agileProcess.pdf page at www.objectmentor.com.

[AHO] Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan. "The AWK Programming Language", Addison-Wesley, 1988.

[ALU] Deepak Alur, John Crupi, Dan Malks, "Core J2EE Patterns - Best practices and Design Strategies", Prentice Hall, 2001.

[ALE] Christopher Alexander, "A Pattern Language: Towns/Buildings/Construction", Oxford Univ. Press (1977).

[BEC] Richard A. Becker, John M. Chambers, Allan R. Wilks, "The New S Language", Wadsworth & Brooks/Cole, 1988.

[BCS] See "The Challenges

of Complex IT Projects" at /BCS/News/PositionsAndResponses /Positions/complexity.htm of www.bcs.org

[BER] Craig A. Berry, John Carnell, Matjaz B. Juric, Meeraj Moidoo Kunnumpurath, Nadia Nashi, Sasha Romanosky, "J2EE Design patterns Applied", Wrox Press (2002).

[BOX] George E. P. Box, William G. Hunter, J. Stuart Hunter, "Statistics for Experimenters", Wiley (1978). Section 2.4 central limit theorem indicates that error due to many random variables tend to approach a normal distribution when the number of variables get larger. One could regard the variability of results from a manufacturing process as a kind of measurement with contribution of many random variables.

[BUS] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, "A System of Patterns - Pattern oriented Software Architecture", Wiley, 1996.

[CHR] Dennis Christenson, Steel Huang, "A Code Inspection Model for Software Quality Management and Prediction", paper presented at IEEE Global Telecommunication Conference & Exhibition, GLOBCOM'88, Hollywood, Florida, Nov. 28-Dec. 1, 1988.

[CMM] /cmm/cmm.html page at www.sei.cmu.edu.

[CMM-2] See "Key Practices of the Capability Maturity Model, version 1.1" by Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, Marilyn Bush, Technical Report CMU/SEI-93-TR-025, Feb. 1993, at /cmm/obtain.cmm.html page of www.sei.cmu.edu

[COC] Adrian Cockcroft, Richard Pettit, "Sun Performance and Tuning - Java and the Internet", second edition, Prentice Hall, 1998.

[COR] "Statistics Over IT Projects Failure Rate", from http://www.it-cortex.com/Stat_Failure_Rate.htm

[COV] Stephen R. Covey, "The 7 Habits of Highly Effective People", Simon & Schuster Fireside Book, 1989. Lots of good tips in management, such as knowing where you are going, first thing first, how to work with people.

[CRA] Warren Craycroft, "Spiraling In: A Medical Monitor Case Study", http://www.projectconnections.com.

[CSI] Mihaly Csikszentmihalyi, "Flow: The Psychology of Optimal Experience", Harper & Row, 1990.

[EXT] Extreme Programming introduction at http://www.extremeprogramming.org. For extreme programming rules and practices see http://www.extremeprogramming.org/rules.html.

[FAR] Many good links and info from Dave Farthings software project management web page. See /pages/staff/dwfarthi/projman.htm at www.comp.glam.ac.uk

[FIS] Roger Fisher and Scott Brown, "Getting Together - Building Relationships as We Negotiate", Penguin Books, 1988.

[FIS-2] Roger Fisher, William Ury, Bruce Patton, "Getting to YES", second edition, Penguin Books, 1991.

[FOW] Martin Fowler, "Patterns of Enterprise Application Architecture", Addison-Wesley, 2003.

[GAM] Erich Gamma, Richard Helm, Ralph

Johnson, John Vlissides, "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

[GRA] Robert B. Grady, Deborah L. Caswell, "Software Metrics: Establishing A Company-Wide Program", Prentice-Hall, 1987.

[GRA-2] Eugene L. Grant, Richard S. Leavenworth, "Statistical Quality Control", sixth edition, McGraw-Hill (1988). Various control charts are described in Part One. P.60 - taking average of subgroup is important, as for subgroup as small as four or five, the average will tend to distribute normally even if the population itself is not distributed like a normal distribution.

[HAT] Erik Hatcher, Steve Loughran, "Java Development with Ant", Manning, 2003.

[HAY] John R. Hayes, "The Complete Problem Solver", The Franklin Institute Press, 1981. See Satisfying: a non-optimizing approach, p.156.

[INM] W. H. Inmon and L. J. Friedman, "Design review Methodology for a Data Base Environment", Prentice-Hall, 1982.

[J2E] The viewgraphs of core J2EE patterns are from the page /blueprints/corej2eepatterns/Patterns/index.html at java.sun.com. J2EE web site is at http://java.sun.com/j2ee/.

[LEY] Frank Leymann, Dieter Roller, "Production Workflow - Concepts and Techniques", Prentice Hall, 2000.

[LIB] Don Libes, "Exploring Expect", OReilly & Associates, 1995.

[LUT] Mark Lutz, David Ascher, "Learning Python" Second Edition, O'Reilly (2004)

[MCG] Michael E. McGill, "American Busi-

ness and the Quick Fix", Henry Holt and Company, 1988.

[MUS] John D. Musa, Anthony Iannino, Kazuhira Okumoto, "Software Reliability - Measurement, Prediction, Application", McGraw-Hill, 1987. For data of system T1, see p. 202 and 305.

[NET] .NET framework homepage at http://www.microsoft.com/net/

[NOR] Peter Norvig, see the article "Design Pattern in Dynamic Programming" at http://norvig.com/design-patterns/

[OPE] Here are some popular open source web sites - www.gnu.org, jakarta.apache.org, www.jboss.org, www.sourceforge.net.

[ORF] Robert Orfali, Dan Harkey, "Client/Server Programming with Java and CORBA", second edition, Wiley, 1998.

[OSI] See "The Myths of Open Source" at Open Source Initiative web site - http://www.cio.com/archive/030104/open.html

[OUS] John K. Ousterhout, "Tcl and the Tk Toolkit", Addison-Wesley, 1994. Two good web sites about Tcl/Tk - http://www.tcl.tk/ and http://wiki.tcl.tk/

[OUS2] An article by John K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century" at http://home.pacbell.net/ouster/scripting.html

[PED] Samuele Pedroni & Noel Rappin, "Jython Essentials", O'Reilly, 2002.

[PRO] Story at projectconnections.com based upon "Open Silicon Takes ASICs Off the Beaten Path", March 29, 2004, EE Times.

[RAT] IBM Rational Unified Pro-

cess web page is located at http://www-306.ibm.com/software/awdtools/rup/

[RAY] http://www.raytheon.com/feature/cmmi/, see remarks by Raytheon's Chairman and CEO Dan Burnham.

[ROB] Thomas G. Robertazzi, "Computer Networks and Systems - Queueing Theory and Performance Evaluation", second edition, Springer, 1994. See section 2.2 for M/M/1 queueing system.

[RUS] Stuart Russell, Peter Norvig, "Artificial Intelligence - A Modern Approach", second edition, Pearson Education (2003)

[SAV] Rob Savoye, document for DejaGnu testing framework at /software/dejagnu/manual/dejagnu.pdf.gz page of www.gnu.org

[SCH] Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, "Pattern-oriented Software Architecture: Patterns for Concurrent and Networked Objects", Vol. 2, Wiley (2000).

[SHA] Mary Shaw, David Garlan, "Software Architecture", Prentice Hall (1996).

[SIX] See Motorola Univ. site where six sigma was invented - http://mu.motorola.com/, another good site is http://www.6siga.com/.

[SOF] Some software six sigma web sites - http://www.softwaresixsigma.com/index.htm, http://main.isixsigma.com/.

[STE] Stephen Stelting, Olav Maassen, "Applied Java Patterns", Prentice Hall, 2002.

[TIG] Open source software engineering web site at http://www.tigris.org/

[VAS] David Vaskevitch, "Client/Server Strategies - A Survival Guide for Corporate

Reengineers", IDG Books, 1993.

[VEN] Bill Venners, "Inside the Java Virtual Machine", 2nd edition, McGraw-Hill (2000).

[YEH] H. T. Yeh, "Software Process Quality", McGraw-Hill (1993).

[YEH-2] H. T. Yeh, "Re-engineering a Software Development Process for Fast Delivery - Approach & Experiences", Proceedings in First International Conference on the Software process, p. 106, IEEE Computer Society Press, 1991.

[YEH-3] H. T. Yeh, M. A. Harding, P. F. Sun, "The Use of Lognormal Distribution to Portray Quality Improvement Trends", paper presented at the 5th Annual Conference on Software Quality & Productivity, Washington, D.C., March 1989.

[YEH-4] Raymond Yeh, Stephanie Yeh, "The Art of Business", May 2004.

About The Author

I was born in China but grew up in Taiwan. I have a life long interest in science and studied physics with degrees from National Taiwan University (B.S.) and University of Illinois (Ph.D.) After graduation, I taught physics and did research at State University of New York at Buffalo for several years. Then I worked at Oak Ridge National Laboratory for fusion research till the late 1970. After that I joined AT&T Bell Laboratories for almost twenty years till retirement. At AT&T I worked on many projects with various roles over the years - software development, process quality and project management. I wrote a book on "Software Process Quality" (McGraw-Hill) in 1993. After retirement, I still do a little training and consulting work on software project and process management.

I am married to Susan Ting and have two wonderful daughters, Emily and Frances. I learned Chinese cooking as a new hobby and routinely cooked during week days before my wife's recent retirement. I also like to hike and travel and Hawaii is my favorite place for vacation. I also like to visit US National Parks. The picture

here shows me (right) and my brother (left), Dr. Raymond Yeh, in the beautiful Zion National Park, May 2003. I also like to read and keep up with the latest development in science and technology. I am fascinated by the rapid ascendancy of science and technology over the last few hundred years and their great impact on human society. It's such a great show I feel very lucky to be born during this age. I hope to get to understand what is consciousness in my life time. I enjoy classical music and dabble a little in composing music for Chinese classical poems. Recently, I learned to self-publish through services at lulu.com and published several books there: "Five Willows Guy" (translation of Chinese poems), "Follow Your Blessings" (essays about simple living), and "Converse With

Spring Wind" (collection of my poems). All my e-books there could be downloaded for free at http://people.lulu.com/users/index.php?fHomepage=101324