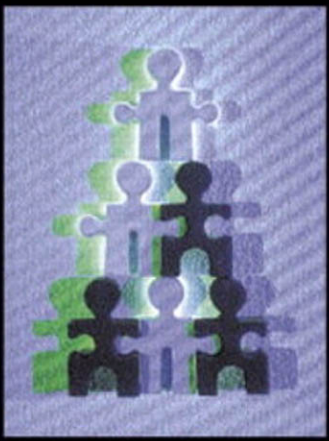
SEI SERIES IN SOFTWARE ENGINEERING

Introduction to the Team Software Process℠



TSPi℠

Watts S. Humphrey

Introduction to the Team Software Process℠

Carnegie Mellon
Software Engineering Institute

The SEI Series in Software Engineering represents a collaboration between the Software Engineering Institute of Carnegie Mellon University and Addison-Wesley to develop and publish a body of work on selected topics in software engineering. The common goal of the SEI and Addison-Wesley is to provide the most current software engineering information in a form that is easily usable by practitioners and students.

For more information point your browser to www.awprofessional.com/seiseries

---

Dennis M. Ahern, et al., *CMMI® SCAMPI Distilled.* ISBN: 0-321-22876-6

Dennis M. Ahern, et al., *CMMI® Distilled, Second Edition.* ISBN: 0-321-18613-3

Dennis M. Ahern, et al., *CMMI® Distilled: A Practical Introduction to Integrated Process Improvement, Third Edition*. ISBN: 0-321-46108-8

Christopher Alberts and Audrey Dorofee, *Managing Information Security Risks.* ISBN: 0-321-11886-3

Julia H. Allen, et al., *Software Security Engineering: A Guide for Project Managers*. ISBN: 0-321-50917-X

Len Bass, et al., *Software Architecture in Practice, Second Edition.* ISBN: 0-321-15495-9

Marilyn Bush and Donna Dunaway, *CMMI® Assessments.* ISBN: 0-321-17935-8

Carnegie Mellon University, Software Engineering Institute, *The Capability Maturity Model.* ISBN: 0-201-54664-7

Mary Beth Chrissis, et al., *CMMI®, Second Edition.* ISBN: 0-321-27967-0

Paul Clements, et al., *Documenting Software Architectures.* ISBN: 0-201-70372-6

Paul Clements, et al., *Evaluating Software Architectures.* ISBN: 0-201-70482-X

Paul Clements and Linda Northrop, *Software Product Lines.* ISBN: 0-201-70332-7

Bill Curtis, et al., *The People Capability Maturity Model®.* ISBN: 0-201-60445-0

William A. Florac and Anita D. Carleton, *Measuring the Software Process.* ISBN: 0-201-60444-2

Brian P. Gallagher, et al., *CMMI®-ACQ: Guidelines for Improving the Acquisition of Products and Services*. ISBN: 0321580354

Suzanne Garcia and Richard Turner, *CMMI® Survival Guide.* ISBN: 0-321-42277-5

Hassan Gomaa, *Software Design Methods for Concurrent and Real-Time Systems.* ISBN: 0-201-52577-1

Elaine M. Hall, *Managing Risk.* ISBN: 0-201-25592-8

Hubert F. Hofmann, et al., *CMMI® for Outsourcing.* ISBN: 0-321-47717-0

Watts S. Humphrey, *Introduction to the Personal Software Process℠.* ISBN: 0-201-54809-7

Watts S. Humphrey, *Managing the Software Process.* ISBN: 0-201-18095-2

Watts S. Humphrey, *A Discipline for Software Engineering.* ISBN: 0-201-54610-8

Watts S. Humphrey, *Introduction to the Team Software Process℠.* ISBN: 0-201-47719-X

Watts S. Humphrey, *Winning with Software.* ISBN: 0-201-77639-1

Watts S. Humphrey, *PSP℠: A Self-Improvement Process for Software Engineers.* ISBN: 0-321-30549-3

Watts S. Humphrey, *TSP℠—Leading a Development Team*. ISBN: 0-321-34962-8

Watts S. Humphrey, *TSP℠—Coaching Development Teams.* ISBN: 0-201-73113-4

Robert C. Seacord, *Secure Coding in C and C++.* ISBN: 0-321-33572-4

Robert C. Seacord, *The CERT® C Secure Coding Standard*. ISBN: 0321563212

Jeannine M. Siviy, et al., *CMMI® and Six Sigma: Partners in Process Improvement*. ISBN: 0321516087

Richard D. Stutzke, *Estimating Software-Intensive Systems.* ISBN: 0-201-70312-2

Sami Zahran, *Software Process Improvement.* ISBN: 0-201-17782-X

Introduction to the Team Software Process℠

Watts S. Humphrey
with Support Tool by
James W. Over

Addison-Wesley

An imprint of Addison Wesley Longman, Inc.

Reading, Massachusetts • Harlow, England • Menlo Park, California
Berkeley, California • Don Mills, Ontario • Sydney
Bonn • Amsterdam • Tokyo • Mexico City

Software Engineering Institute

The SEI Series in Software Engineering

The publisher offers discounts of this book when ordered in quantity for special sales. For more information, please contact:

Computer and Engineering Publishing Group
Addison Wesley Longman, Inc.
One Jacob Way
Reading, Massachusetts 01867
(781) 944-3700

ISBN 0-201-47719-X
Text printed in the United States at Demand Print Center in Old Tappan, New Jersey.
11th printing, June 2009

IN MEMORY OF MY FATHER, WATTS S. HUMPHREY (1896–1968)

He provided critical support at a difficult time in my life.

This page intentionally left blank

FACULTY FOREWORD

The increasing complexity of software development and the demand by industry for better-qualified and better-prepared software engineers means software development curricula must provide students with knowledge and experience related to the practice of software engineering. The Embry-Riddle Aeronautical University Industrial Advisory Board[1] has identified the following issues as critical for the preparation of entry-level software engineers:

communication (both oral and written)

ability to work as part of a team

front-end part of software development (requirements and high-level design)

professional attitude toward work

knowledge and skills in using software processes

computing fundamentals

breadth of knowledge (ability to learn new technologies)

In my eight years of teaching an introductory software engineering course, I have tried to provide to students an overview of the full "life-cycle" development of software and to have them work as part of a team. I have tried real projects, toy projects, small-team development, large-team development, extensive tool use, almost no tool use, emphasis on product issues, and emphasis on process issues. Until recently, I have had only limited success. Most of the time, I have tried to do

[1]The Advisory Board members come from several organizations, including Boeing, Harris, Lockheed-Martin, Motorola, and the Software Engineering Institute.

too much, with the result that both the students and the teacher ended up frustrated and disappointed.

This past year, I used a draft of *Introduction to the Team Software Process*℠ (TSPi) and had the best success in my experience. Although the TSPi is no silver bullet, it has had a dramatic effect in improving our delivery of software engineering education. The TSPi shows both teachers and students what to do, how to do it, and when to do it. This book includes all the required TSPi materials: the scripts, forms, and instructions for almost all aspects of student-team software development. It does an excellent job of explaining and motivating the TSPi activities, and it provides a complete description of each team role. It offers commonsense advice on how to handle team management problems, and it specifies quantitative techniques for planning, tracking, and assessing performance and quality.

Although the book includes an initial statement of requirements for two projects, the TSPi is flexible enough to handle a variety of projects of modest size. The TSPi could be adjusted for a maintenance project, used in a requirements/design course, or adapted to just about any team software activity. For example, we are now using the TSPi in our senior design course to develop a product for a real customer. The TSPi incorporates an incremental development methodology that provides a sound software development strategy and an excellent pedagogy. In the first increment (using a simple set of requirements), students learn the TSPi process and get comfortable working with a team. In the subsequent one or two increments, the teams can use their previous experience to improve their performance.

Although there is much to be gained by using the TSPi in a software engineering project course, students must have prior experience with the Personal Software Process (PSP)℠, either through a previous course or by self-study. At Embry-Riddle, we introduce our students to the PSP in their freshman programming courses; this provides sufficient preparation for study and use of the TSPi. The TSPi course requires a great deal of time on the part of both teacher and students. In several attitudinal surveys (one at the end of each increment), students overwhelmingly endorsed the TSPi. A common complaint concerned the collection and recording of data—although most admitted they understood its importance.

In summary, if you have struggled with how to deliver a quality software engineering project course, I strongly encourage you to look at the TSPi. It provides the guidance, direction, and support for teaching students the practice of software engineering and preparing them for the workplace.

Thomas B. Hilburn,
Embry-Riddle Aeronautical University

STUDENT FOREWORD

To help in preparing this student foreword, Professor Tom Hilburn selected three students from his first TSPi course at Embry-Riddle. Two of these students had worked as coops in industry, and all three were team leaders for their teams. Tom asked these students what they would say to their classmates when asked about this course; the following paragraphs are excerpts from what they wrote.

Why learn TSPi? Why use it when it takes more time to fill out all the forms than to do the project itself? These questions have very good answers. When you learn something new, you do not want to try something so big that you cannot handle it. You need to start small to get the hang of what is going on. That is how you learn TSPi. You start with a small project that could probably be done without the process. Once you have mastered the TSPi, you realize that it is a necessity. Although the TSPi is not needed for most school assignments, it will be needed for larger industrial projects. This is where a process like TSPi will help everyone understand how to do the project. So remember that you are just learning the process, and the benefits of this knowledge will not show up until you are faced with a project that cannot be cranked out at a terminal.

Most computer science students learn how to develop programs individually. In their team projects, they would love to have a structured process to help them in their next team experience. Team interaction is a whole new aspect of the software process. Many issues, such as communication, trust, motivation, problem-solving, commitment, dedication to quality, balancing workloads, allocating roles, feelings of camaraderie, authority issues, and learning about your teammates and how they work, are new issues that are factored into this team process. The TSPi provides advice for handling many of these common issues. Students need to adapt the process to their particular team situation and to learn how to handle other issues that are not addressed.

Some key things we learned were the necessity of good communication and trust among team members and how to work with people we do not know. Most of the team wanted to complete the tasks early, to provide plenty of time for quality reviews. Team members either volunteered to take on additional tasks when the work suited their skills or helped someone who was overloaded. This type of teamwork motivated us not to let the team down and to help out when and where we could. Especially in the first stage of the project, our team was more motivated to produce the product than to follow the process. In the second stage, the engineers could see the value of the forms and planning. This provided more motivation to follow the process.

We had few motivation or commitment problems, but when we did, we had to only briefly discuss them. The hardest thing was to be objective in decisions and dealings with other team members. If we had not communicated as well or as often as we did, the project would have been half as effective, if that. We learned the responsibilities of teamwork, how the roles interrelate, and the importance of helping each other. Also, we learned that if you put forth the effort, the team will be willing to help you. The TSPi gives students a great head start for what they will be dealing with in industry. Ideally, curricula will be better structured in the future to provide such courses.

A word on project management: TSPi requires a good deal of management skill and competence. A wide variety of tools are available to the team through the TSPi, but it is up to the team leader to set the stage for the project. As team leader, I tried several approaches to project management; most worked, and at least one did not. As leader, you must be willing to put in as much time as your teammates and to "get down in the dirt" and help when needed. A leader must be a motivator—by word, by deed, or by both. Give your team the room to work and be creative in their designs, but guide them through the process and foster the teamwork that will prove so valuable in the last days, when problems and failures are most likely. In the short time available, be sure to involve the team in creating the plan, to get everyone's input for each step, and to watch the team's schedule and determine a plan of action if it begins to slip.

The team leader needs to plan for some situations before they occur. Maybe a team member does not contribute or is lost due to injury or illness. Both these situations have happened on my projects. Such problems can tear a team apart and ruin its effectiveness, or they can pull a team closer together and increase its productivity. It all depends on how the situation is handled. That is when you must be willing to lead. Enjoy the team experience, get to know your team, and try to help them motivate themselves and the rest of the team. Work hard and plan. Learn how to use a spreadsheet to save time in number crunching. Keep the lines of communications open, and meet often. Remember that when everything seems to hit you at once, there is a way through it. "Never quit, never die!"

Celeste Berry, Ryan Hoppes, Marc Lovelace
Embry-Riddle Aeronautical University

PREFACE

This book is for students and engineers who have already learned and, preferably, applied the Personal Software Process (PSP)[SM]. You may have learned the PSP in a graduate or senior-level course[1] or in an earlier introductory course.[2] Alternatively, you may be a practicing engineer who seeks guidance on how to use the PSP in an industrial team environment. In any case, when you have learned the PSP, you have the background to use the methods and practices in this book.

After you have learned the PSP, you may need guidance on applying it to the many tasks of the software process. This is the principal role of the Team Software Process (TSP)[SM]: to provide a framework for using sound engineering methods while developing software.

There is a great deal to say about teamwork, and this book covers the basic elements. TSPi (the introductory Team Software Process) introduces team concepts and walks you through the steps of building teams and working on a team. Note, however, that this text is designed for an introductory course and does not cover all the material that you will need to use the TSP for larger-scale industrial projects.

[SM]Personal Software Process and PSP are service marks of Carnegie Mellon University.

[1]The advanced PSP course is taught from my text, *A Discipline for Software Engineering,* Addison-Wesley (1995).

[2]The beginning PSP course uses my book *Introduction to the Personal Software Process,* also from Addison-Wesley (1997).

[SM]Team Software Process and TSP are service marks of Carnegie Mellon University.

How TSPi Helps Engineers

This book teaches engineers about software development teamwork. TSPi provides a structured set of steps, shows engineers what to do at each step, and demonstrates how to connect these steps to produce a completed product. TSPi also provides two interesting and reasonably challenging project exercises. Each is at the same time small enough to be completed in a few weeks and large enough to simulate a typical small project. When capable engineers follow the guidance provided in this book, they will invariably produce a finished working product.

In the suggested TSPi strategy, teams develop a product in two or three cycles. In the first cycle, teams build a small working product kernel. With each succeeding cycle function is added to this base. This strategy demonstrates the benefits of using data from a prior project to plan a new project. Also, by taking new roles for each cycle, engineers will have two or three quite different experiences in just one project. After several development cycles, engineers will have had a broad exposure to teaming methods, and they are likely to continue using the TSPi methods on their own.

Why TSPi Courses Are Needed

Because project courses have proven to be effective in preparing students for software engineering careers, a growing number of universities now offer them. These courses are often oversubscribed. Students seek material that applies to their future jobs, and they see team courses as meeting this need. After graduation, students and employers report that software project courses are useful preparation for work in industry.

There is now a large body of experience with team project courses.[3] Although many of these courses have been successful, three problems are common. First, the students often attempt projects that are too large. Second, they frequently concentrate on the product and ignore the process. Finally, one or more team members are disruptive. Although TSPi cannot prevent all these problems, it provides guidance on how to avoid or mitigate them.

[3]See, for example, A.T. Berztiss, "Failproof Team Projects in Software Engineering Courses," *Frontiers in Education Conference* (IEEE, 1997); D.H. Hutchens, "Using Iterative Enhancement in Undergraduate Software Engineering Courses," *SIGCSE '96*; T.J. Scott, "Team Selection Methods For Student Programming Projects," *8th CSEE, '95*; and J.E. Tomayko, "Carnegie Mellon Software Development Studio: A Five-Year Retrospective," *Proceedings of the Ninth Conference on Software Engineering Education* (IEEE Computer Society Press, 1996).

To make effective use of curriculum time, team software courses should be carefully structured and based on proven project experience. *Without a defined process or a structured team framework, engineers must figure out for themselves how to run their projects.* Without this process and structure, these groups must learn team-building and teamwork basics through an often painful trial-and-error process. This is both expensive and unnecessary because teamwork principles are well known and straightforward.

TSPi guides engineers in effective teamwork methods. It does this by walking them through a team-building process and then using a measured and defined framework for developing products. Assuming that the engineers are PSP-trained, they can follow the TSPi scripts and use the TSPi support tool to plan and manage their work.[4] Following TSPi makes engineers' projects much more efficient and permits them to concentrate on learning about software engineering rather than spend an excessive amount of time on team-building and team management issues.

TSPi provides defined team roles that are allocated among the team members. Each role specifies what is expected and when and how each task is to be done. When all team members know what they and everyone else should do, they are in a better position to work effectively as a team. If a team member does not do his or her job, the other team members will know it, and they can deal with the problem. When teams cannot solve interpersonal problems themselves, they are told to call on their instructor or manager for help. The Instructor's Guide for this book suggests methods for handling many common teamwork issues.

When student team members have explicit roles and the role responsibilities are clearly defined and visible, instructors can provide fairer and more specific grades. Each student can then be rated on individual performance as well as on the overall team's results. Not only does this approach motivate better performance, but it is also a fairer way to grade team courses.

The Organization of This Book

This book is designed to lead teams through the TSPi process. Following the first two introductory chapters (Part I), the chapters in Part II walk teams through a complete development cycle. The text explains the process scripts and gives examples of the completed TSPi forms.

Part III provides detailed descriptions of the TSPi team-member roles: team leader, development manager, planning manager, quality/process manager, and support manager. After reading the chapter on your personal role, you can use these TSPi role scripts for reference while working on the project.

[4]The TSPi support tool, the TSPi Instructor's Guide, and other support supplements for this text are described on the Supplements page in the back of this book.

At the start of the TSPi course, each student completes an INFO form (see Appendix F) describing his or her interests and background. The instructor uses this information to divide the class into five-engineer teams and to assign initial roles to the team members. If one or two teams have four or six members, the instructor must make some role adjustments. All the roles must be assigned, and each engineer should have at least one role. For a four-engineer team, the support manager role should be distributed among the team members. For a six-engineer team, the quality/process manager role should be split into two: the quality manager and the process manager.

With the teams selected and roles assigned, the teams start their projects and report on their progress. At the end of each development cycle, the engineers assess the team's overall performance as well as that of each individual role. With this information, the instructor can evaluate the work and better assign team-member roles for the next development cycle. If necessary, the instructor may make some team membership changes, but, unless there are serious problems, teams should be kept together throughout the course.

Using Standard, Predefined Problems

Although TSPi will work for almost any project, this book provides two standard, predefined problems that are designed to meet the needs of a wide variety of courses. Although there could be advantages to using actual customer problems, this practice is not recommended for three reasons. First, courses have firm and unvarying schedules. Although most customers will initially agree to a fixed time scale, few customers know how long it really takes to develop software. Also, because beginning engineers do not generally know how to manage projects on firm schedules, the chances of project failure are high. This problem is compounded by the fact that actual customer requirements are notoriously vague and unstable, leading to frequent changes and extensive delays.

The second reason to use a standard, predefined exercise is that a teamwork course should be designed to teach specific lessons. Although one goal of the project should be to build a working product, the principal course objective should be to demonstrate the benefits of using proven software engineering methods. With an actual customer problem, the first priority must be to satisfy the customer. As the requirements change or the customer takes time to answer questions, the work will slip. As the schedule gets compressed, teams often concentrate on finishing the product and ignore the process. Unfortunately, the principal lesson often learned from such courses is how *not* to develop software.

The third reason to use a standard, predefined problem is that it permits each team to compare its performance with that of other teams. With several implemen-

tations of the same problem, all the teams can participate in the class evaluations. Each team can describe its approach and answer questions about its design, implementation, and test choices. This process graphically shows the effectiveness of various development approaches and provides a body of reference data for evaluating future teams.

Although there are advantages to using standard predefined exercises, they do not expose students to some important issues. For example, without actual experience, it is hard to appreciate the confusion and imprecision of customer need statements. Struggling with vague and changing requirements is an important experience, but it can be taught best in a course that concentrates on the requirements process. The approach recommended here is to first teach effective teamwork and process methods and then, in later courses, focus on the complex issues of larger-scale development projects.

Suggestions for Instructors

This book can be used in several ways. The principal use is in a full one- or two-semester team course. In this configuration, TSPi is used to develop a single product such as either of the two described in Appendix A. A one-semester course would take two or three cycles, whereas a two-semester course would use three or more cycles to build a larger product or a full-function version of the Appendix A products. Depending on the scale of the job, the various process steps could be expanded or reduced. Three course options are shown in Figures P.1, P.2, and P.3.

For each development cycle in Figure P.1, the team plans and tracks its work and completes a full miniproject, including requirements, design, code, and test. At the end of each development cycle, the team assesses team and role performance, and the instructor reassigns the team roles. In a three-cycle project, the engineers gain experience with three essentially complete projects and three different team roles. They also have data from each cycle and can see how their experience from one cycle can be used for the next one.

This book can also be used for teamwork exercises in other courses. Small projects could be done in a single cycle of three to seven weeks. A short requirements cycle could take three or four weeks, whereas a design cycle would take four or five weeks. The shortest initial full development project would take six or seven weeks. Figure P.2 shows a several-week team project to develop a set of requirements in a requirements course. Similarly, a design project might be configured as in Figure P.3. The text can also be used for courses that run on a quarter system. Whereas the full three-cycle course takes 15 weeks, a two-cycle course can be done in 11 weeks, and a one-cycle development project would take 7 weeks.

Cycle Week	Cycle 1	Cycle 2	Cycle 3
1	Course introduction, review		
2	Launch, strategy		
3	Plan		
4	Requirements		
5	Design		
6	Implementation		
7	Test		
8	Postmortem	Launch, strategy, plan	
9		Requirements, design	
10		Implementation, test	
11		Postmortem	Launch, strategy, plan
12			Requirements, design
13			Implementation, test
14			Test, documentation
15			Postmortem and evaluation

FIGURE P.1 THE THREE-CYCLE TSPi COURSE

Cycle Week	Cycle 1
1	Launch, plan
2	Requirements
3	Requirements
4	Postmortem

FIGURE P.2 A SHORT TSPi REQUIREMENTS PROJECT

Cycle Week	Cycle 1
1	Launch, plan
2	Requirements
3	Design
4	Design
5	Postmortem

FIGURE P.3 A SHORT TSPi DESIGN PROJECT

In any of these course configurations, the standard TSPi scripts guide the students through forming their teams and planning and implementing their projects. Unless a team has already had experience with a full TSPi course, they will probably not complete any project cycle in less than three or four full weeks. The reason is that it takes time for new team members to learn the process and to figure out how to work together as a team. This is why the first TSPi cycle is planned for seven weeks even though the same work would take only four weeks in a subsequent cycle.

Preparation for This Course

The principal prerequisite for this course is completion of a full PSP course. This PSP course can be either a graduate or an introductory course. If the students took the PSP several semesters ago, they should have used the PSP in their intervening courses. If they have not, they will need a brief refresher lecture or two about PSP planning, data gathering, and quality management. Also, students who have little or no experience using the PSP will almost certainly need careful monitoring and support throughout the team course.

Before attempting a team project, students should have a background in software design and software requirements. Exposure to configuration management, project management, and software testing is also helpful. The students must also be fluent in the programming language and the tools they will use.

Acknowledgments

In writing a book, I often become so immersed in the material that I find it hard to see many of the problems that could trouble first-time readers. This is the principal reason that I seek informed reviewers. I have been particularly fortunate with this book, both because many people were willing to help and because their broad range of backgrounds enabled them to make many helpful suggestions. I particularly appreciate the help and support of Susan Brilliant, Dan Burton, Bob Cannon, Audrey Dorofee, Pat Ferguson, Marsha Pomeroy Huff, Mark Klein, Susan Lisack, Rick Long, Steve Masters, Mark Paulk, Bill Peterson, Bill Pollack, Dan Roy, Jeff Schwalb, Girish Seshagiri, Steve Shook, Laurie Williams, Ralph Young, Dave Zacharias, and Sami Zahran. Julia Mullaney was a great help in combing through the manuscript to find problems and inconsistencies in the manuscript and text. I also wish to thank my brother Philip Humphrey for his continued support and informed comments on much of the teamwork material.

I am also particularly indebted to Tom Hilburn and Iraj Hirmanpour at Embry Riddle Aeronautical University. Both of them have been long-term supporters of my PSP and TSP work. Tom has also taught team courses using the manuscript for this book. The data from his first course provided much of the material for the examples in the text.

As we at the Software Engineering Institute (SEI) have gained experience with the PSP and TSP, the importance of tool support has become increasingly clear. Jim Over has developed a marvelous tool to support TSPi teams, and he has adapted it specifically to support this book. He has also provided many helpful comments on both the process and the text. For that I am deeply grateful.

Again, I am indebted to Peter Gordon and Helen Goldstein and the professional staff at Addison-Wesley. Their help and guidance were invaluable in making the book a reality. Finally, I must again thank Barbara, my wife, for her continued support and good-natured encouragement through yet another book.

I dedicate this book to the memory of my father, Watts S. Humphrey, whose trust, confidence, and enthusiastic support helped and sustained me through my formative years. One of my very earliest memories is of failing first grade. In those days, they did not know about learning disabilities, but my father instinctively knew that I could learn, given proper guidance and instruction. He insisted that I had not flunked, but the school had, so he moved our family to a town where my brothers and I could attend a school that would give me individual instruction. I was extraordinarily fortunate to have had such a father, and I am deeply grateful for his help and support. Although he died many years ago, I still miss him.

Watts S. Humphrey *Sarasota, Florida*

CONTENTS

This page intentionally left blank

PART ONE

Introduction

Part I of this book describes the introductory Team Software Process (TSPi) and explains why it is needed and how it works. Chapter 1 covers the benefits that you can expect from a TSPi course and the principles behind TSPi's design and structure. Chapter 2 describes teams: what they are and what makes them work. The material in Chapter 2 also includes a discussion of teamwork problems and describes how the TSPi can help in handling these problems. Chapter 2 briefly deals with the people-related issues of teams and teamwork. These topics are covered in considerably more detail in the chapters in Part IV.

This page intentionally left blank

1

TSPi Overview

Most industrial software is developed by teams. Thus, to be an effective engineer, you need to be able to work on a team. If you have good sense and a willingness to cooperate, you have the basic equipment to be a successful team member. Teamwork, however, is more than just getting along. Teams must plan their projects, track their progress, and coordinate their work. They also must agree on goals, have a common working process, and communicate freely and often.

To meet aggressive schedules and produce high-quality products, practiced teamwork is essential. However, practiced teamwork requires experience and calls for a specific set of skills and methods. This textbook and its accompanying course provide a comprehensive introduction to team software development. The approach is to expose you to realistic teamwork problems and to give you practical teamwork experience. With the experience gained from this course, you will be prepared to participate in a large-scale industrial software project.

This chapter describes the introductory Team Software Process (TSPi), the principles behind the TSPi design, and the overall structure and flow of the TSPi process. It also describes why TSPi is needed and discusses the benefits you can expect from a TSPi course.

1.1 What Is TSPi?

TSPi is a defined framework for a graduate or upper-level undergraduate course in team software engineering. It provides a balanced emphasis on process, product, and teamwork, and it capitalizes on the broad base of industrial experience in planning and managing software projects. TSPi guides you through the steps of a team software project course, and it shows you how to apply known software engineering and process principles in a teamwork environment. Assuming that you have already learned the Personal Software Process (PSP)[SM], TSPi will show you how to plan and manage a team project. It also defines roles for you and your teammates. When everyone on the team has an explicit role with clearly defined responsibilities, you can see what you are supposed to do at each step of the process. By following the TSPi process, you will get practical experience with proven engineering and teamwork methods.

The TSPi design is based on the Team Software Process (TSP)[SM], an industrial process for teams of as many as 20 engineers who develop or enhance large-scale software-intensive systems. Because the TSP is designed for large projects that often take several years to complete, it is a larger and more complex process than you will need. TSPi is thus a reduced-scale version of TSP. It does, however, retain the same basic concepts and methods. After using TSPi, you will find the TSP quite familiar and easy to use.

Why Engineering Teams Need a Process

Merely giving a group of engineers a job does not automatically produce a team. The steps required to build a team are not obvious, and new teams often waste a substantial amount of time handling teamwork mechanics. They must figure out how to work together as a team, how to define the job they need to do, and how to devise a strategy for doing the work. They must allocate the tasks among team members, coordinate each of these tasks, and track and report on their progress. Although these team-building tasks are not trivial, they are not very difficult. There are known methods for doing every one of them, and you and your teammates need not reinvent these methods for yourselves.

Teams don't just happen, and superior team performance is not an accident. Although skilled members and a defined process are essential, teams are more than a collection of talented individuals. To build and maintain effective working relationships, you need common goals, an agreed plan of action, and appropriate leadership. You also need to understand one another's strengths and weaknesses, support your teammates, and be willing to call for help when you need it.

[SM]Personal Software Process and PSP are service marks of Carnegie Mellon University.
[SM]Team Software Process and TSP are service marks of Carnegie Mellon University.

TSPi will also improve your productivity. Although the early planning and team-forming steps may seem to take a great deal of time, they are an essential part of doing a team project. It is a little like the huddle in a football game: experienced teams first agree on the play and each team member's role in it. If football teams didn't huddle, they would do a lot of running around, but they wouldn't win many games.

1.2 TSPi Principles

The TSPi is based on the following four basic principles.

1. Learning is most effective when you follow a defined process and get rapid feedback. The TSPi scripts and forms provide a defined, measured, and repeatable framework for team software engineering. The TSPi provides rapid performance feedback because the team produces the product in several short development cycles and evaluates results after each cycle.
2. Productive teamwork requires a combination of specific goals, a supportive working environment, and capable coaching and leadership. The project goal is to build a working product. The TSPi provides the supportive environment, one of your team members will be the team leader, and the instructor provides the coaching.
3. When you have struggled with actual project problems and have been guided to effective solutions, you will appreciate the benefits of sound development practices. Without the precise guidance of the TSPi, however, you could waste considerable time in defining your own practices, methods, and roles.
4. Instruction is most effective when it builds on the available body of prior knowledge. There has been a great deal of experience with software teams and software team courses. TSPi builds on this foundation.

1.3 The TSPi Design

There are many ways to design a process. In the case of TSPi, there were seven principal design decisions.

1. Provide a simple framework that builds on the foundation of the Personal Software Process (PSP).
2. Develop products in several cycles.

3. Establish standard measures for quality and performance.
4. Provide precise measures for teams and students.
5. Use role and team evaluations.
6. Require process discipline.
7. Provide guidance on teamwork problems.

The following sections discuss each of these topics in turn.

Provide a Simple Framework That Builds on the Foundation of the PSP

The purpose of a process is to help you do a task, such as developing a product or learning how to do a team project. When the process is too complex, you spend so much time figuring out what to do that you can easily lose sight of the objective. On the other hand, if the process is too simplistic, it does not provide enough guidance. You must then invent your own process as you go along. This approach can lead to wasted time, costly mistakes, and even a complete project failure.

Although TSPi has many forms and scripts, most of them are similar to ones you have already used with the PSP. Thus, if you have been PSP-trained, you will need to learn the overall TSPi approach, but you already understand the "language" of the TSPi. If you have not yet been PSP-trained, however, the TSPi process will likely seem overpowering. This is one reason that prior PSP training is a prerequisite for a TSPi course.

Even with PSP training, you must learn a number of new tasks and activities. However, with the role assignments, TSPi permits you to focus on the elements that you personally need to learn. The TSPi roles specify who does each of the team's planning, tracking, quality, support, and leadership tasks. The roles also reduce the amount of material you must understand at the outset. You can concentrate on your specific role tasks and not worry about the other role responsibilities, at least not yet.

During a multicycle TSPi project, try to get experience with several team roles. This experience will provide you the broadest exposure to the TSPi process, and it will give you a deeper understanding of more aspects of team software development. It will also help you to better appreciate your personal interests and abilities.

Develop Products in Several Cycles

In a full TSPi course, you will complete two or three development cycles in one semester. Each of these cycles includes a full requirements, design, implementation, and test development process. In the first cycle, you build a minimum-function subset of the ultimate product. By quickly getting this kernel product running, you can be sure to have a working product at the end of the course. You will also have

a solid base for each of the subsequent cycles. This strategy is also more likely to identify design, performance, or usability problems at the beginning of the project when they are easiest to fix.

The second TSPi cycle builds on the results of the first cycle. You might change team roles, adjust the process, or use more disciplined quality methods. At the end of the second cycle, you will have data on two complete projects. Assuming that there is time, you could repeat these cycles for a third and even a fourth time. After two or more cycles, you will feel like an old hand at the development business. You will have clear and convincing evidence of what works best for your team, and you will have the confidence to continue using these methods in practice.

Establish Standard Measures for Quality and Performance

Measurements are an essential part of doing consistently high-quality work. The PSP provides the basic measures and measurement skills you need to measure and evaluate the quality of your work. Until you have lived through a project that uses these measures, however, it is hard to appreciate their value. TSPi shows you how to interpret the PSP measures and how to apply them to a project.

TSPi requires that you and your teammates set personal as well as team goals. Although this task may not seem important, a key part of an engineering education is learning to establish ambitious but attainable goals. The TSPi emphasis on goals and measures helps you to see the benefits of quality measurements and the value of project planning and tracking. Sound planning and tracking also help you to manage your work on a daily basis.

Provide Precise Measures for Teams and Students

With the TSPi measures, your performance and that of your teammates will be obvious. Although it is important for everyone to try hard and to do good work, a common team problem is that some members contribute much less than their fair share of the work.

Although the principal purpose of the TSPi measures is to help you do better work, these measures also make your personal performance visible to your teammates. This, however, is the nature of teamwork: Everyone knows what everyone else is doing. Therefore, if you do not make a reasonable effort, you can expect comments from your teammates.

Use Role and Team Evaluations

Some groups like to use peer evaluations, and others do not. The TSPi provides a peer evaluation capability that can be used if the instructor so desires. The advantage of peer evaluations is that the students are best informed about the team's and

one another's performance. If they can be persuaded to make an honest evaluation, the instructor will be best informed and best able to give fair and equitable grades.

Because students are naturally reluctant to evaluate their peers, the TSPi calls for team and role evaluations. The idea is to evaluate how each role was performed and not how the people behaved. Although a role evaluation could be read as a judgment of the person who performed that role, the TSPi emphasis is on evaluating how the process worked and not on how the people performed.

Require Process Discipline

It is hard for software engineers to consistently do disciplined personal work. There are three reasons for this.

1. Software engineering has no tradition of disciplined personal performance. Thus, neither students nor working engineers have role models to emulate.
2. The software process does not impose a natural discipline on engineers. Software development differs from hardware engineering, in which engineers release designs to a factory for volume production. Hardware production engineers must review and accept a released design before they commit to manufacture a product on a schedule, for a cost, and at a scheduled production rate. With software, we have no factory and no production engineers. Therefore, software engineers must discipline themselves.
3. Consistently disciplined work in any field requires high standards and competent support. That is why professionals in the sports and performing arts have coaches, trainers, conductors, and directors. In industry, some accomplished managers have learned to fulfill this coaching role; in academic courses, the instructor acts as the team coach. Unfortunately, however, few people know how to be effective coaches, and coaching is not widely practiced in the software field.

With TSPi, the instructor will require you to follow the process and to gather the data. You must complete the forms and analyze and use the data. If you do not do this, you will not reap the full benefits of the TSPi course.

Provide Guidance on Teamwork Problems

Even in the best-run projects it is common to have teamwork problems. You and your teammates have different roles, and each of these roles has its own objectives. When these objectives conflict, disagreements are likely. In fact, it would be surprising if you did not have some disagreements with your teammates. Don't assume that these are personality problems, however, unless the conflicts become

unresolvable. Initially, the best assumption is that your teamwork issues are caused by process problems.

Occasionally, students have problems working on a team. This is not only because they are inexperienced but also because they have backgrounds or personalities that make teamwork difficult. However, with guidance and support, most engineers can be effective team members. The most powerful force for resolving team problems is peer pressure. Most people are concerned about their peers' opinions and are eager to be respected and to fit into the group. Given time and proper coaching, most bright and motivated engineers can learn how to be fully effective team members.

If your team members cannot work together effectively, ask the instructor for help. The instructor, in cooperation with your team, can often help you to overcome the problems. If even this does not work, it may be necessary to remove some member from the team. Although this is a sensitive step, it is usually not a problem if the rest of the team asks the instructor to remove the problem member. For more guidance on these topics, consult Chapters 11, 16, and 17.

1.4 TSPi Structure and Flow

Figure 1.1 shows how the TSPi uses multiple development cycles to build a final product. Cycle 1 starts with a launch, in which the instructor describes the overall product objectives. The team then follows TSPi through its seven process steps: strategy, planning, requirements, design, implementation, test, and postmortem. In cycle 2, the engineers repeat the same steps, this time enhancing the base product produced in cycle 1. If there is time, they can add further enhancements in subsequent cycles.

The Cyclic Development Strategy

When you start a cyclic development strategy, the best plan is to begin with the smallest viable product version. In deciding the size and content of each cycle, you should consider the following constraints.

1. Each cycle should produce a testable version that is a proper subset of the ultimate product.
2. Each cycle should be small enough to be readily developed and tested in the available time.
3. When combined, the cycle products should produce the desired final product.

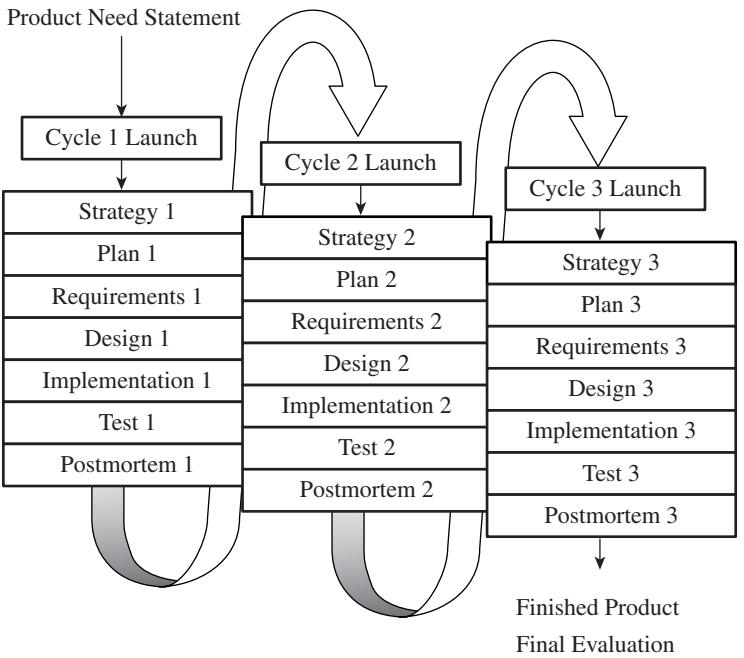


FIGURE 1.1 TSPi STRUCTURE AND FLOW

The TSPi starts by having teams produce a development strategy. Start by picking the smallest reasonable base to develop during the first cycle. Then estimate the sizes of the functions you plan to add in each subsequent cycle. This approach almost guarantees that you will complete a working initial subset of the product. With the data from this initial cycle, you can accurately plan what to add in each subsequent cycle. Don't defer too much function to cycles 2 and 3, however, because the course schedule provides less time for these later cycles.

1.5 The TSPi Process

The development script (DEV) in Table 1.1 shows the overall TSPi flow. Each script step is supported by one or more detailed scripts. Each of these scripts is described in a chapter devoted to that process step. All the TSPi scripts are also included in Appendix D. Table 1.1 also shows a typical TSPi script structure. Here,

TABLE 1.1 TSPi DEVELOPMENT: SCRIPT DEV

Purpose		**To guide a team through developing a software product**
Entry Criteria		• An instructor guides and supports one or more five-student teams. • The students are all PSP-trained (*Discipline for Software Engineering* or *Introduction to the Personal Software Process*). • The instructor has the needed materials, facilities, and resources to support the teams. • The instructor has described the overall product objectives.
General		The TSPi process is designed to support three team modes. 1. Develop a small- to medium-sized software product in two or three development cycles. 2. Develop a smaller product in a single cycle. 3. Produce a product element, such as a requirements document, a design specification, a test plan, and so on, in part of one cycle. Follow the scripts that apply to your project and mode of operation.
Week	**Step**	**Activities**
1	Review	• Course introduction and PSP review. • Read textbook Chapters 1, 2, and Appendix A.
2	LAU1	• Review course objectives and assign student teams and roles. • Read textbook Chapter 3, Appendix B, and one of Chapters 11–15.
	STRAT1	• Produce the conceptual design, establish the development strategy, make size estimates, and assess risk. • Read textbook Chapter 4.
3	PLAN1	• Produce the cycle 1 team and engineer plans. • Read textbook Chapter 5 and Appendix C.
4	REQ1	• Define and inspect the cycle 1 requirements. • Produce the system test plan and support materials. • Read textbook Chapter 6 and the test sections of Chapter 9.
5	DES1	• Produce and inspect the cycle 1 high-level design. • Produce the integration test plan and support materials. • Read textbook Chapter 7.
6	IMP1	• Implement and inspect cycle 1. • Produce the unit test plan and support materials. • Read textbook Chapter 8.
7	TEST1	• Build, integrate, and system test cycle 1. • Produce user documentation for cycle 1. • Read textbook Chapter 9.

TABLE 1.1 (continued)

Week	Step	Activities
8	PM1	• Conduct a postmortem and write the cycle 1 final report. • Produce role and team evaluations for cycle 1. • Read textbook Chapters 10, 16, 17, and 18.
	LAU2	• Re-form teams and roles for cycle 2. • Read the rest of textbook Chapters 11–15.
	STRAT2, PLAN2	• Produce the strategy and plan for cycle 2. • Assess risks.
9	REQ2	• Update the requirements and system test plan for cycle 2.
	DES2	• Produce and inspect the cycle 2 high-level design. • Update the integration plan for cycle 2.
10	IMP2	• Implement and inspect cycle 2, produce unit test plan.
	TEST2	• Build, integrate, and system test cycle 2. • Produce user documentation for cycle 2.
11	PM2	• Conduct a postmortem and write the cycle 2 final report. • Produce role and team evaluations for cycle 2.
	LAU3	• Re-form teams and roles for cycle 3.
	STRAT3, PLAN3	• Produce the strategy and plans for cycle 3. • Assess risks.
12	REQ3	• Update the requirements and system test plan for cycle 3.
	DES3	• Produce and inspect the high-level design for cycle 3. • Update the integration plan for cycle 3.
13	IMP3	• Implement and inspect cycle 3, produce unit test plans.
	TEST3	• Build, integrate, and system test cycle 3.
14	TEST3	• Produce and review the user manual for the finished product. • Review and update the user manual for usability and accuracy.
15	PM3	• Conduct a postmortem and write the cycle 3 final report. • Produce role and team evaluations for cycle 3. • Review the products produced and the processes used. • Identify the lessons learned and propose process improvements.
Exit Criteria		• Completed product or product element and user documentation. • Completed and updated project notebook. • Documented team evaluations and cycle reports.

the left column contains a sequence number indicating the order of the script steps, or, in this case, the week when that script is scheduled. The second column briefly names the topic of that script section, and the third column contains the descriptive text for each script step.

Every script starts with a brief statement of the overall purpose of the activity. This activity could be, for example, to develop a requirements document, produce a design, or conduct a test. Every script also has entry and exit criteria. These specify what you need to have done before you start the script and what you should have accomplished by the time you finish. Following the entry criteria, there is a "General" section that provides general information about the script. Finally, the numbered script rows are the activities that you are to follow in enacting the script.

1.6 The Textbook Structure and Flow

This textbook has a preface, four main parts, several appendixes, and an index. The Preface discusses where and how this textbook should be used. Part I provides an introduction to the TSPi process and explains what it is and why it is structured the way it is. Part II walks you through the TSPi process, one major step at a time. In Part III you will find a chapter on each of the five standard TSPi roles. Part IV offers guidance on dealing with some of the issues you will likely face in working on a software engineering team.

The seven textbook appendixes contain basic reference materials: the exercise need statements, descriptions of the configuration management and inspection processes, the process and role scripts, the TSPi forms, and the TSPi standards and specifications. Inside the front and back covers you will also find a glossary of the special terms used in the text. Finally, at the back of the book is a complete subject index.

It is most important that you read the textbook chapters on each of the process steps before you attempt that step. The suggested order for reading the textbook chapters and appendixes is given in the DEV script (Table 1.1).

1.7 Summary

The four basic TSPi principles are as follows.

1. Learning is most effective when students follow defined and repeatable steps and get rapid feedback on their work.

2. Productive teamwork requires a defined team goal, an effective working environment, and capable coaching and leadership.
3. When students are exposed to the problems of realistic development projects and then guided to effective solutions, they gain a better appreciation of the value of sound engineering.
4. Instruction is most effective when it builds on the available body of prior engineering, scientific, and pedagogical experience.

Starting from these four principles, the TSPi design involves seven choices.

1. Provide a simple framework that builds on the foundation of the PSP.
2. Develop products in several cycles.
3. Establish standard measures for quality and performance.
4. Provide precise measures for teams and students.
5. Use role and team evaluations.
6. Require process discipline.
7. Provide guidance on teamwork problems.

The TSPi process follows a cyclic development strategy. By starting with a small set of initial functions, the team can quickly complete a working first version of the product. When members have produced this initial version, they can better plan and develop the second cycle product. If there is time for a third cycle, this learning process is further reinforced. The cyclic development strategy is much like the processes followed by successful large-scale software development groups.

2

The Logic of the Team Software Process

This chapter discusses the introductory Team Software Process and explains how and why it works. It also defines teams, explains how they work, and discusses some common teamwork problems.

Much has been written about teamwork, and there are many examples of good and bad teams. This chapter cannot possibly cover all this material, but it hits the highlights. For further information about teamwork issues, see Chapters 16 and 17. This chapter is only an introduction. As you use the TSPi, you should read each of the book chapters as you encounter the topics it covers. The contents of this chapter are as follows.

- Why projects fail. A principal problem for software teams is learning how to handle pressure. A poor or ineffective response to pressure is often the cause of project failure. We open the chapter with a discussion of pressure and how TSPi can help teams to handle the normal pressures of software work.
- Common team problems. A number of studies of student teams have identified their most common problems. We also describe some of the problems that the TSPi process is designed to address.
- What is a team? Before discussing teams, we must agree on what a team is.
- Effective teams. Some teams are more effective than others. This chapter section discusses the conditions and characteristics that differentiate successful teams from all the others.

- How teams develop. Effective teams don't just appear; they usually develop. This chapter section summarizes the process through which effective software teams develop, either by chance or through a deliberate team-building process.
- How TSPi builds teams. The next topic is a brief review of the steps TSPi uses to build effective teams.

2.1 Why Projects Fail

When software projects fail, it is generally because of teamwork problems and not technical issues. DeMarco [DeMarco 88, page 2] says that

> The success or failure of a project is seldom due to technical issues. You almost never find yourself asking 'has the state of the art advanced far enough so that this program can be written?' Of course it has. If the project does go down the tubes, it will be non-technical, human interaction problems that do it in. The team will fail to bind, or the developers will fail to gain rapport with the users, or people will fight interminably over meaningless methodological issues.

One significant "people" problem is the inability of software teams to handle pressure, especially the pressure to meet an aggressive development schedule. Often, teams respond to this pressure by taking shortcuts, using poor methods, or gambling on a new (to them) language, tool, or technique.

Excessive pressure can be destructive. It causes people to worry and to imagine problems and difficulties that may not be real. Rather than help you to cope in an orderly and constructive way, pressure causes worry about many unknown (and often phantom) issues. And sometimes pressure can cause you to act as if the phantom issues were real. This behavior can have untold consequences for your project, your organization, and even your self-esteem.

When your team knows how to handle the pressure of a tight schedule, you can feel the difference. Before starting a job, you generally don't know precisely what is involved. But after you make a plan and get started, you feel relieved. This is true even if the job is larger than you thought. The reason you feel relieved is that you are now dealing with a known problem rather than an unknown worry.

Handling Pressure

Pressure is something that you feel. For example, you may need to do a task whether or not you think you can do it. The greater the need and the more doubt you have about your ability to do the task, the greater the pressure. Because pressure is internally generated, you have the power to manage it yourself, but first you must

find the source of the pressure and then figure out how to deal with it. The apparent source of pressure in software projects is the need to meet a tight schedule. This schedule could come from management, your instructor, or your peers.

The real source of pressure, however, is ourselves. It comes from our natural desire to accomplish what our managers, instructors, or peers want. When this pressure is coupled with normal self-doubts about our ability to perform, it can become destructive. This is particularly true for new software teams that have not yet learned how to handle the normal challenges of their projects. Teams need to know how to work efficiently and to produce quality products, especially when they are under intense schedule pressure.

By guiding teams through a strategy and planning process, the TSPi shows teams how to handle pressure. They analyze the job, devise a strategy for doing the work, estimate the sizes of the products they will build, and then make a plan. Because unrealistic schedules are the principal cause of software project problems, the TSPi helps teams manage their projects more effectively. When they are able to manage their work, teams are much more likely to do a quality job.

2.2 Common Team Problems

Although working on teams can have tremendous advantages, there can also be problems. Studies have found that the most common problems for student teams concern leadership, cooperation, participation, procrastination, quality, function creep, and evaluation [Pournaghshbanb].

Ineffective Leadership

Without effective leadership, teams generally have trouble sticking to their plans and maintaining personal discipline. Although effective leadership is essential, few people are natural leaders. Most of us need to develop our leadership skills and to get practice using them. Until engineers have seen effective leadership in action, however, they often don't know what skills to practice.

Failure to Compromise or Cooperate

Occasionally one or more team members may not be willing or able to work cooperatively with the team. Although this does not happen often, teams need to deal with this problem when it arises. Peer pressure can often resolve such problems, but if a person continues to be intractable you should discuss the problem with your instructor.

Lack of Participation

Team members have different skills and abilities as well as different motivations, energy, and levels of commitment. This means that every member makes a different level of contribution to the team's performance. In fact, the variation among the members' contributions generally increases with increasing group size [Shaw, page 202].

Although some degree of variation in participation is normal, it is important that all team members strive to meet the team's goals. If it becomes clear that someone is not making a serious effort, team spirit generally suffers. Nothing can be more disruptive than to have some people in a group openly getting away with something. Lee Iacocca calls this the equality of sacrifice: "If everybody is suffering equally, you can move a mountain. But the first time you find someone goofing off or not carrying his share of the load, the whole thing can come unraveled" [Iacocca, page 230].

Procrastination and Lack of Confidence

Some teams do not set deadlines or establish goals and milestones. Others set deadlines they never meet. Such teams generally don't track performance and often fail to make decisions in a timely or logical way. They take excessive time to get started, and they drift through their projects rather than attacking them. These problems generally stem from one or more of the following three things: inexperienced leadership, a lack of clear goals, or the lack of a defined process and plan.

Poor Quality

Quality problems can come from many sources. Examples are a superficial requirements inspection, a poorly documented design, or sloppy implementation practices. When teams do not use personal reviews or team inspections, they usually have quality problems, resulting in extensive testing, delayed schedules, long hours, and an unsatisfactory final product.

Function Creep

During product design and implementation, engineers often see ways to improve their products. These well-intentioned modifications are hard to control because they originate from a legitimate desire to produce a better result [Robillard, page 89]. This problem is particularly difficult because there is no clear dividing line between the functions that stem from interpretations of the requirements and those that are true additions to the requirements.

Ineffective Peer Evaluation

Experience has shown that peer evaluation can be invaluable for student teams [Scott, page 302]. However, students are often reluctant to grade their teammates and rarely do so with complete candor. As a result, students often feel that the grading in team courses is not entirely fair, particularly to the highly motivated students. This perception can cause competition among team members and can reduce the willingness of team members to fully cooperate.

2.3 What Is a Team?

There are many definitions of teams. The one I like best is by Dyer [Dyer, page 286]:

> A team consists of
>
> (a) at least two people, who
>
> (b) are working toward a common goal/objective/mission, where
>
> (c) each person has been assigned specific roles or functions to perform, and where
>
> (d) completion of the mission requires some form of dependency among the group members.

Team Size

Teams can be of almost any size from two to dozens or even hundreds of people. In practical situations, however, teams are most effective when they develop close relationships among all the members. This is most likely when the teams are small and when the members develop a network of interdependencies. In industry, team size is generally limited by management span of control. Although some projects can be very large, generally there are smaller subgroups of 20 or fewer people, each of them working under the direction of a supervisor or manager. These subgroups form the close-knit teams that the TSP and TSPi are designed to support.

Student teams typically range from about four to 12 students, depending on class size and faculty preferences. Although there have been few studies of the effects of software team size, my experience has been that teams of four to eight engineers are likely to be the most effective. With fewer than four members, there are not enough people to properly handle all the team role assignments. With teams of more than eight members, it is harder for the team to develop the close relationships needed for teams to jell. TSPi is designed for teams of five students,

although it can be used with modest changes for teams of four or six students. For teams of other sizes, more role adjustments are required but the other scripts and forms apply.

The Jelled Team

When design and development groups work together smoothly and efficiently, we call them *jelled* teams. DeMarco and Lister, in their marvelous book *Peopleware,* talk about the jelled team [DeMarco 87, page 123]:

> A jelled team is a group of people so strongly knit that the whole is greater than the sum of the parts. The production of such a team is greater than that of the same people working in unjelled form. Just as important, the enjoyment that people derive from their work is greater than what you'd expect given the nature of the work itself.

Basic Teamwork Conditions

Not all groups are teams. There are three basic conditions that must be met for a group to operate successfully as a team [Cummings, page 627; Dyer, page 286; Mohrman, page 279].

1. The tasks to be done are clear and distinct; that is, the job for the team is explicitly defined, the work is meaningful to the team, and the group knows what it must do.
2. The team is clearly identified; the members know the scope of the group, who is in it, and who is not. Everyone on the team is known to the others, everyone's work is visible, and everyone knows everyone else's team role.
3. The team has control over its tasks; members know what to do, how to do it, when to do it, and when they are finished. The members know that they are responsible for the work, and they control the processes they use. They also have the capability to do the job, and they know that no one else is charged with doing it.

2.4 Building Effective Teams

To build effective teams, you need more than just the right kinds of tasks and working conditions. The team must have an important job to do and must be in an environment that supports teamwork. The team must face an aggressive challenge and must be encouraged to plan and manage its own tasks. These needs are met by

providing the team with four additional kinds of support: cohesion, goals, feedback, and a common working framework.

Team Cohesion

Cohesion refers to the tight knitting of the team members into a unified working group that physically and emotionally acts as a unit. Members of highly cohesive groups communicate freely and often. Although they need not be good friends, they work closely together and respect and support one another. In less-cohesive groups, members tend to function as individuals. They have trouble compromising and do not have common values and goals. Cohesive teams, however, share a common physical space, spend a lot of time together, and supportively cooperate and interact during these times together.

Challenging Goals

Goals are also a critical element of the jelled team. First, these goals must be specific and measurable. Studies show that teams that have measurable goals are consistently more effective than those that do not [Mohrman, page 176]. Examples of such goals are detailed plans, performance targets, quality objectives, schedule milestones, and so on. And each of the team members must accept these goals as his or her own.

Second, the team's goals must represent a significant challenge [Katzenbach, page 3]. No team jells without a performance challenge that is meaningful to those involved. Although good personal chemistry and the desire to become a team can foster teamwork values, these characteristics alone will not automatically produce a jelled team.

Finally, the goals must be tracked and progress visibly displayed so that the team members can see how they are progressing toward their goals.

Feedback

Goal tracking and feedback are critically important. Effective teams are aware of their performance and can see the progress they are making toward their goals [Stevens, page 515]. In a study of air defense crews, those with frequent and precise feedback on goal performance improved on almost every criterion. This compares with the stable, unimproving performance of crews that did not get feedback [Dyer, page 309].

The team members must also be able to distinguish their personal performance from that of the team as a whole. When they cannot do this, team performance generally suffers. Called *shirking,* this results from people expending less personal effort when the results of their work are not apparent to the rest of the team. The basic

cause of shirking is the lack of a team member's personal commitment to the team's common goals. The presence of one or more shirkers generally prevents a team from jelling. Shirking is not primarily a measurement problem, but precise measures generally reveal such problems so that the team can deal with them.

Common Working Framework

Whereas the team's goals must be challenging and clear, the path to achieving them must also be clear: "Team members need to see how to achieve the goal and know what is expected of them" [Shaw, page 388]. This means that all the team members must feel that the tasks are achievable, must understand their roles and responsibilities, and must agree on how to accomplish them. They must know

What tasks must be done?

When?

In what order?

By whom?

In summary, to jell, teams must be cohesive, have challenging goals, get frequent performance feedback, and have an agreed-upon process or framework for doing the work.

2.5 How Teams Develop

Teams don't just happen; they generally develop over time. This development can happen by luck, or it can result from a conscious team-building process. At the outset, most teams start with individuals who have diverse goals. As teams jell, the members come to accept a common set of team goals. When they do, these goals take on a special significance. Even though the goals may be arbitrary, the team members will pursue them with enormous energy. They do this not because of the nature of the goals but rather because the goals are important to the team.

How Teams Jell

The first step in creating a jelled team is for the team members to converge on a common understanding of the product that they intend to build. This forms a starting point for the team to develop its goals and to make a plan. After the team defines its goals, the members agree on a strategy and a plan for developing the product.

Studies have found that knowledge work can be viewed as an iterative process that begins with several engineers, each of whom has a different understanding of what he or she is to do. Then, through a series of steps, they converge on a common viewpoint and result [Mohrman, page 52]. As the team members increase their common understanding of the product to be built, they also converge on a common approach for doing the work. Throughout this convergence process, the team is gradually becoming a more cohesive unit.

In the beginning, the engineers are not sure what the product will look like or how they should build it. Although they cannot yet agree on the product or the full development process, they can usually agree on the current unknowns and on how to clarify them. Then they proceed in iterative steps: identifying the confusions and disagreements, agreeing on how to resolve them, and resolving them. They then move to a more detailed level, identify additional confusions and disagreements, and resolve them. As they converge on a common understanding, they simultaneously converge on the details of the intended product and the processes for producing it.

What many engineers find surprising is that conflict, confusion, and disagreement are natural parts of this convergence process. It is how the team identifies the issues to work on, and it is what generates the creative process that we call design.

The TSPi supports this jelling process by walking teams through a launch procedure that addresses the conditions required for jelled teams.

2.6 How TSPi Builds Teams

Most small groups can become effective teams by focusing on the basic techniques of team development. These techniques help teams to build the understandings and relationships they need to work together and to support one another. The TSPi guides teams through the team-building steps to set goals, select roles, establish plans, and maintain communication among the team members.

Goals

As teams begin to jell, they first define and accept a set of common goals. The TSPi helps the members accept the team goals by having all team members participate in defining them. Because goal setting is difficult, particularly for new teams, the TSPi defines an initial set of team and team-member goals (see Chapters 11–15 for the role specifications). For the second and subsequent TSPi cycles, teams should review and adjust these goals based on their experience with the first development cycle.