SEI软件工程丛书 · 影印版

团队软件过程

Introduction to the Team Software Process

瓦茨 · S · 汉弗莱 [Watts S. Humphrey] 著

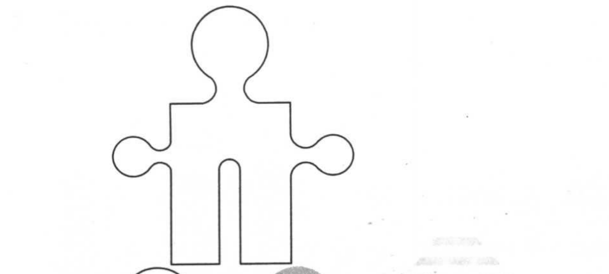
清华大学出版社

SEI 软件工程丛书 · 影印版

团队软件过程

Introduction to
the Team Software Process

瓦茨 · S · 汉弗莱 [Watts S. Humphrey] 著



清华大学出版社

（京）新登字 158 号

内 容 简 介

本书介绍了团队软件过程（TSP）基础知识，适用于软件开发项目经理、程序员和一般编程爱好者在开发软件时参考，也可作为高等学校计算机软件工程课程的参考教材使用。

图书在版编目（CIP）数据

团队软件过程 Introduction to the Team Software Process /（美）汉弗莱著. —影印本.—北京：清华大学出版社，2002
（SEI 软件工程丛书）
ISBN 7-302-06141-6
Ⅰ. 团… Ⅱ. 汉… Ⅲ. 软件工程 Ⅳ. TP311. 5

中国版本图书馆 CIP 数据核字（2002）第 097400 号

出 版 者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
http://www. tup. com. cn
http://www. tup. tsinghua. edu. cn
责任编辑：尤晓东
印 刷 者：清华大学印刷厂
发 行 者：新华书店总店北京发行所
开 本：787×960 1/16 **印张**：31. 5 **插页**：2 **字数**：508 千字
版 次：2002 年 12 月第 1 版 2002 年 12 月第 1 次印刷
书 号：ISBN 7-302-06141-6/TP・3672
印 数：0001～3000
定 价：49. 00 元

出 版 说 明

1984 年，美国国防部出资在卡内基·梅隆大学设立软件工程研究所(Software Engineering Institute，简称 SEI)。SEI 于 1986 年开始研究软件过程能力成熟度模型（Capability Maturity Model，CMM），1991 年正式推出了 CMM 1.0 版，1993 年推出 CMM 1.1 版。此后，SEI 还完成了能力成熟度模型集成（Capability Maturity Model Integration，简称 CMMI）。目前，CMM 2.0 版已经推出。

CMM 自问世以来备受关注，在一些发达国家和地区得到了广泛应用，成为衡量软件公司软件开发管理水平的重要参考因素，并成为软件过程改进的事实标准。CMM 目前代表着软件发展的一种思路，一种提高软件过程能力的途径。它为软件行业的发展提供了一个良好的框架，是软件过程能力提高的有用工具。

SEI 十几年的研究过程和成果，都浓缩在由 SEI 参与研究工作的资深专家亲自撰写的 SEI 软件工程丛书（SEI Series In Software Engineering）中。

为增强我国软件企业的竞争力，提高国产软件的水平，经清华大学出版社和三联四方工作室共同策划，全面引进了这套丛书，分批影印和翻译出版，这套丛书采取开放式出版，不断改进，不断出版，旨在满足国内软件界人士学习原版软件工程高级教程的愿望。

清华大学出版社

2002 年 8 月

总　　序

周伯生

美国卡内基•梅隆大学软件工程研究所（CMU/SEI）是美国联邦政府资助构建的研究单位，由美国国防部主管。他们确认，为了保证软件开发工作的成功，由软件开发人员、软件采办人员和软件用户组成的集成化团队必须具有必要的软件工程知识和技能，以保证能按时向用户交付正确的软件。所谓“正确的”就是指在功能、性能和成本几个方面都能满足用户要求且无缺陷；所谓“无缺陷”就是指在编码后对软件系统进行了彻底的穷举测试修复了所有的缺陷，或保证所编写的代码本身不存在缺陷。

CMU/SEI 为了达到这个目的，提出了创造、应用和推广的战略。这里的“创造”是指与软件工程研究社团一起，共同创造新的实践或改进原有的实践，而不墨守成规。这里的“应用”是指与一线开发人员共同工作，以应用、改进和确认这些新的或改进的实践，强调理论联系实际。这里的“推广”是指与整个社团一起，共同鼓励和支持这些经过验证和确认的、新的或改进的实践在世界范围内的应用，通过实践进行进一步的检验和提高。如此循环，往复无穷。

他们把所获得的成就归纳为两个主要领域。一个是倡导软件工程管理的实践，使软件组织在采办、构建和改进软件系统时，具有预测的能力与控制质量、进度、成本、开发周期和生产效率的能力。另一个是改进软件工程技术的实践，使软件工程师具有分析、预测和控制软件系统属性的能力，其中包括在采办、构建和改进软件系统时，能进行恰当的权衡，作出正确的判断和决策。CMU/SEI 通过出版软件工程丛书，总结他们的研究成果和实践经验，是推广这两个领域经验的重大举措。

SEI 软件工程丛书由 CMU/SEI 和 Addison-Wesley 公司共同组织出版，共分 4 个部分：计算机和网络安全（已出版了 2 本著作），工程实践（已出版了 8 本著作），过程改进和过程管理（已出版了 11 本著作），团队软件过程和个体软件过程（已出版了 3 本著作）。前两者属于软件

工程技术实践，后两者属于软件工程管理实践。目前这 4 个部分共出版了 24 本著作，以向软件工程实践人员和学生方便地提供最新的软件工程信息。这些著作凝聚了全世界软件工程界上百位开拓者和成千上万实践者的创造性劳动，蕴含了大量的宝贵经验和沉痛教训，很值得我们学习。

清华大学出版社邀请我和郑人杰教授共同组织 SEI 软件工程译丛编委会。清华社计划首先影印 6 本著作，翻译出版 15 本著作。据我所知，在 Addison-Wesley 公司出版的 SEI 软件工程丛书中，人民邮电出版社已经翻译出版了《个体软件过程》和《团队软件过程》，还拟影印出版《个体软件过程》和《软件工程规范》；电子工业出版社已经翻译出版了《净室软件工程的技术与过程》、《能力成熟度模型 CMM 1.1 指南》、《能力成熟度模型集成 CMMI》和《软件项目管理》；北京航空航天大学出版社已经翻译出版了《统计过程控制》。这些出版社共计影印 2 本著作，翻译出版 7 本著作。这样，可以预期我国在今年年底共可影印 8 本著作，翻译出版 22 本著作。各个出版社的有远见的辛勤劳动，为我们创造了“引进、消化、吸收、创新”的机遇。我们应该结合各自的实践，认真学习国外的先进经验，以大大提高我国软件工程的理论和实践水平。

在这套丛书中，特别值得一提的是，在过程工程领域被誉为软件过程之父的 Humphrey 先生所撰写的《软件过程管理》、《技术人员管理》、《软件工程规范》、《个体软件过程》、《团队软件过程》和《软件制胜之道》等 6 本著作，将于今年年内全部翻译出版，其中《软件过程管理》、《技术人员管理》、《软件工程规范》、《个体软件过程》和《软件制胜之道》等 5 本著作亦已经或将于今年年内影印出版。

《软件过程管理》是软件过程领域的开创性著作，是为软件公司经理和软件项目经理撰写的。用这本书提出的原理来指导软件开发，可以有效地按照预定进度得到高质量的软件，同时还可了解如何持续进行过程改进。美国 CMU/SEI 按照这本书提出的原理开发了能力成熟度模型，在国际上得到绝大多数国家的认可和广泛采用，是改进软件过程能力的有力武器。在信息技术迅速发展和企业激烈竞争的今天，能否持续改进过程往往决定企业的命运。

作为一个软件经理，在改进组织的能力之前，首先必须明确绝大多

数软件问题是由管理不善所引起的。因此，要改进组织的性能，首先需要改进自己的管理模式。同时还要认识到软件开发是一项智力劳动，需要拥有掌握高技能和忘我工作的技术人员。因此，有效的软件管理需要充分注意技术人员的管理。

《技术人员管理》这本著作就是为达到这个目的而撰写的。高质量的技术工作要求没有差错，这就要求人们高度专心和高度献身。因此要求人们对他所从事的工作不仅具有高度的责任感，而且具有浓厚的兴趣和高度的热忱。在当前知识经济群龙相争的今天，一个能激励人们进行创造性工作的领导群体，是众多竞争因素中最重要的因素。本书提供了大量的实用指南，可用来有效地改进工程人员、经理和组织的性能。

Humphrey 先生还认为这本书特别适合于在我国工作的软件经理。我国是一个人口大国，拥有大量能干的知识分子，而且信息领域的劳动力价格比国际市场的价格要低，因此吸引了许多国家到我国来投资。但若不提高人员的素质，不在产品质量和进度方面也狠下功夫，就不能在这方面持续保持优势。

《软件工程规范》是为编程人员撰写的。它精辟地阐述了个体软件过程（PSP）的基本原理，详尽地描述了人们如何来控制自己的工作，如何与管理方协商各项安排。在软件工程界，这本著作被誉为是软件工程由定性进入定量的标志。目前在世界范围内，有成千上万的软件工程技术人员正在接受有关 PSP 的培训，以便正确地遵循 PSP 的实践、开发和管理工作计划，在他们承诺的进度范围内，交付高质量的产品。

《软件制胜之道》这本著作描述了团队软件过程的基本原理，详尽地阐述了在软件组织中如何应用 PSP 和 TSP 的原理以及它所能带来的效益。此外，虽然 CMM 同样适用于小型组织，但在其他著作中都没有描述如何应用 CMM 于个体或小型团队，这本书填补了这个空白。应该指出，如果一个组织正在按照 CMM 改进过程，则 PSP 和 TSP 是和 CMM 完全相容的。如果一个组织还没有按照 CMM 改进过程，则有关 PSP 和 TSP 的训练，可以为未来的 CMM 实践奠定坚实的基础。

在软件工程技术实践方面目前共出版了 10 本著作，其中《用商业组件构建系统》、《软件构架实践》和《软件构架评估——方法和案例研究》等 3 本著作详尽地阐述了软件构架的构建、实践和评估。鉴于是否有一个稳定的软件构架，对软件的质量和成本影响很大，因此如何获得

一个良好的构架就成为当今软件界研究的重点。我相信这几本著作的出版，将对我国软件构架领域的研究与实践有重要的参考价值。此外，众所周知，计算机与网络的安全问题对信息系统的可靠使用关系极大，《CERT 安全指南——系统与网络安全实践》的出版将会对我国在这一领域的研究和实践起积极的促进作用。《风险管理——软件系统开发方法》、《软件采办管理——开放系统和 COTS 产品》、《项目管理原理》、《软件产品线——实践和模式》和《系统工程：基于信息的设计方法》等 5 本著作，分别从风险管理、软件采办、项目管理、软件产品线以及信息系统设计方法等几个方面阐述了大型、复杂软件系统的开发问题，是有关发展软件产业的重要领域，很值得我国软件产业界借鉴。

目前我们所处的时代是信息化时代，是人类进入能够综合利用物质、能量和信息三种资源的时代。千百年来以传统的物质产品的生产、流通、消费为基本特征的物质型经济，将逐步进入以信息产品的生产、流通、利用和消费为基本特征的知识型经济。在这个历史任务中，建造和广泛应用各类计算机应用系统是其公共特征。计算机软件是计算机应用系统的灵魂，没有先进的软件产业，不可能有先进的信息产业，从而也不可能建成现代化的知识型经济。

我们应该看到，在软件领域中我国在总体上离世界先进水平还有相当大的差距。但是，我们不能跟随他国的脚印，走他人的老路。我们应该抓住机遇，直接针对未来的目标，在软件工程技术和软件工程管理两个方面，注意研究 SEI 软件工程丛书中倡导的原理和方法，联系实际，认真实践，并充分利用我国丰富优秀的人力资源和尊重教育的优良传统，大力培养各个层次的高质量的软件工程人员，使其具有开发各类大型、复杂软件系统的能力。我衷心地预祝清华大学出版社影印和翻译出版这套丛书，在把我国建设成为一个真正现代化的软件产业大国的历史任务中起到推波助澜的作用，并请读者在阅读这些译著时，对这套丛书的选题、译文和编排等方面都提出批评和建议。

周伯生
于北京
2002 年 8 月 18 日

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

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

TSPi Glossary

Term	Definition or Formula	Reference
A/FR	Appraisal to failure ratio. *A/FR* = (appraisal time)/(failure time)	102
Appraisal time	Time spent appraising the product, either in personal reviews or inspections.	102
Assemblies	When a program has several parts, it is called an assembly.	70
Balanced plans	Team plans in which every engineer plans to complete his or her personal tasks at the same time. With a balanced plan, the schedule is minimized.	66
Baseline	The collection of documents and other materials that officially represent the product at any point in time.	326
Black-box testing	Testing that is done without knowledge of the program's internal structure.	131
Build	The build process assembles the system's parts for integration and system testing. One such assembly is called a build.	180
Capture-recapture	A method for estimating the size of a population. In TSPi, this method is used to estimate the number of defects remaining in a product after an inspection.	345
CCB	See *configuration control board.*	324
Checklist	A list of items to check in a review. In PSP and TSPi, checklists identify the most likely defects to look for.	339
Configuration change request (CCR)	The official means for transmitting to the CCB a requested change to the product baseline.	324
Configuration control board (CCB)	An engineering committee that reviews and approves the product baseline and proposed changes to it.	324
Configuration item list	The approved list of items to be included in the product or system baseline. It typically includes the product name, when baselined, the owner, and where stored.	331
Configuration management	The total set of activities used to manage the content of the product (the baseline) from the beginning to the end of the development process.	321
Configuration management plan	This plan includes, at a minimum, the configuration identification plan (configuration item list), the configuration control procedures, and the CCB membership.	323
Configuration	The CSR summarizes the status of the configuration status report (CSR) management system at any point in time.	328
Cyclic development	A development process that builds products in steps, each of which produces a working subset of the final product.	6
Defect	A requirements, design, or implementation element that, if not changed, could cause improper design, implementation, test, use, or maintenance.	100
Defect prevention	The process, method, tool, or other actions needed to prevent specific defect types from recurring.	146

TSPi Glossary

Term	Definition or Formula	Reference
Defect profile	A graph of the defect-removal history of a program by phase. It is typically given in defects/KLOC.	100
Defect-prone modules	Those modules in a system that, based on their defect-removal history, are judged likely to have defects remaining.	171
Defect ratio	The ratio of the numbers of defects found in a review phase and a compile or test phase. Low ratios typically indicate low-quality reviews.	101
Defects/KLOC	The number of defects in a program, normalized by the number of KLOC in the program. Defects/KLOC = 1000*(defects found)/(LOC)	100
Development cycle	That development activity that builds one product increment in a cyclic development process.	6, 9
Earned value (EV)	The percent the estimated task hours are of the total project hours. EV is earned only when the task is fully completed.	67
Estimated defects remaining	Using the capture-recapture method, the estimated remaining defects are T = A*B/C, where A is the number of major defects found by one engineer, B the number found by another engineer, and C the number found by both.	345
Facilitator	A person who helps teams hold efficient and effective team meetings. The principal responsibilities are to keep the meeting focused on the agenda and ensure that everyone participates.	204, 212
Failure time	The time spent finding and fixing defects. In PSP and TSP, it is the total time an engineer spends compiling and unit-testing a program.	102
Inspection	A process in which several engineers review a product produced by another engineer to help find defects.	335
Integration test	The test that verifies that the system is properly built, that all the parts are present, and that they function together.	181
Issue	A known problem or concern that must be addressed.	53
KLOC	Thousands of lines of code, or LOC.	85
LOC	Lines of code, as defined in the team's lines-of-code standard.	144
Major defects	These are the problems that, when fixed, would change the executable program.	100
Minor defects	All defects that are not major.	100
Module	Modules are typically composed of multiple objects or functions, and they are the smallest testable program element.	70
Notebook	See *project notebook.*	213
Part	Any part of a system. It could be an object, module, component, product, or subsystem. Parts may also be assemblies of lower level parts.	70
PDF	See *percent defect-free.*	98

(continued inside back cover)

TSPi Glossary

Term	Definition or Formula	Reference
Peer review	See *inspection.*	335
Percent defect-free (PDF)	The percent of a system's parts that have no defects in a specified defect-removal phase.	98
Phase	A process typically has several steps or phases, each one generally described by scripts.	10
Phase yield	The percentage of the defects in a product that are removed during a specified phase. Compile process yield refers to the percentage of the defects injected before the end of compile that are removed during compile. See also *yield.*	106
PIP	The process improvement proposal, a TSPi form.	186
Planned value	The percentage of the total job that is represented by a single task.	67
Prediction interval	The limits within which an estimate is likely to fall. Typical prediction intervals include a percentage, say 95%, of the items being estimated.	350
Process yield	The percentage of the defects injected before a phase that are removed before that phase. Yield before compile refers to the percentage of the defects injected before compile that are removed before compile. See also *yield.*	106
Project notebook	The official record of all the team's estimates, work products, reports, plans, and forms.	213, 214
Quality plan	The planned and actual quality performance of every part and assembly in the system.	97
Rates, inspection and review	One measure of the quality of a review or an inspection. Typically, rates are measured in LOC or pages per hour.	102
Recorder	The person who documents meeting results, principally the decisions and planned actions.	262
Regression test	When programs are modified, functions that were previously tested may no longer work. Regression testing checks for such problems.	182
Reuse	The use of unmodified previously developed program elements in a new program. Typically, reused program elements are taken from a reuse library that is designed specifically for reuse.	54
Review	In the PSP and TSPi, reviews are done by engineers to find defects in their personally produced products.	131
Risk	A problem that may or may not occur.	53
Role	A defined area of responsibility for a team member.	302
Script	A listing of the actions required to accomplish a specific process or portion of a process.	10
SCM	See *configuration management.*	321
SDS	See *software design specification.*	135

TSPi Glossary

Term	Definition or Formula	Reference
Size standard	This specifies how size is to be measured for each product type. It typically includes the LOC counting method, the page standard, and design size measures.	144
Software configuration management	See *configuration management.*	321
Software design specification (SDS)	The program's or system's design, typically including the high-level design, the program architecture, interface definitions, and other important specifications.	135
Software requirements specification (SRS)	A description in the engineers' words of the product they plan to develop, typically reviewed by the customers or users to ensure agreement on the needs.	11
SRS	See *software requirements specification.*	112
System test	A test that stresses the system to determine whether or not it functions properly in all important respects.	181
Task	An element of work in the development plan. In TSPi, the work is typically broken into tasks that can be done in about 10 engineer hours or fewer.	68
Test plan	The test plan defines the tests to be run, the expected test results, the materials needed for testing, and the plan for producing these materials.	169
Unit test	This white-box test verifies that the program structure is correct and that operation is proper with all normal and abnormal variable and parameter values.	381
White-box testing	Testing that is done with knowledge of the program's internal logic and structure.	131
Yield	The percentage of the defects in a program that are removed from the program. See also *phase yield* and *process yield.*	106

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*SM (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 common-sense 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)SM, 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.