Choose Your Weapon Wisely

A handbook for determining the right software development method best suited for your team, client, and project

Justin Rockwood

Carnegie Mellon University

Table of Contents

Abstract

In war, battles can be won or lost depending on the weapons that the armies wield. If the weapon matches the situation, the army can be victorious; otherwise the battle will be lost. It is the same with choosing your "weapon" in a software project. Choosing the right development process can help your project succeed, but choosing a process that doesn't match your particular needs can cause your project to ultimately fail.

In this paper, I have chosen to focus on five of the most popular processes in use today: Rational Unified Process (RUP), Microsoft's Synch and Stabilize (MSS), Team Software Process (TSP), Extreme Programming (XP), and Scrum.

The goal of this paper is twofold. First and foremost, I hope to provide enough impartial information to educate the novice manager or practitioner as to the tradeoffs and inherent strengths and weaknesses of each of the chosen processes. Secondly, I hope to present both novice and seasoned managers and practitioners a mechanism whereby they can determine which processes are best suited to their particular project, team, and customer. This is done by providing a series of questions designed to help them frame their desired goals within the context of specific processes (namely RUP, MSS, TSP, XP, and Scrum). The questions are backed by relative weights which assign numeric values to a series of attributes, such as team dynamics and project type. These relative weights are supported by empirical evidence presented through case studies and current industrial and academic research.

It must be stressed that the purpose of this paper is not to recommend one process over another blindly. Each process has inherent strengths and weaknesses, and despite the claims made by proponents of each process, there is not a single process that works equally well over every single type of project that exists in industry today. Each process must be evaluated and weighed in the context of a specific project in order to determine the best match.

Acknowledgements

Without the help of my advisor, İpek Özkaya, none of this would have been possible. She not only provided invaluable references, advice, and feedback, but she kept me sane throughout the researching and writing of this paper by helping me stay focused on the end goal.

I would also like to acknowledge the help of David Root, my Studio mentor, who convinced me that I should stick with this project and endure to the end.

1 Introduction

1.1 The State of the Art

In the early days of computer systems software was small and nimble and usually written by one sole programmer. The complete application could be understood in its entirety by a single person. Since changes could be quickly and easily accommodated, not a huge amount of planning or design was done before diving into the code. Barry Boehm terms this as the "Code-and-Fix" model, where the programmer writes some code and then thinks about the requirements, design, test, and maintenance later [Boehm 88].

As demand for software increased, so did its size and complexity. Functional programming languages and design techniques started to evolve and the famous waterfall method[1] of software development was born. Additional techniques and processes, such as object-oriented design and languages, were created to cope with the exponential increase in complexity.

Today we are faced with colossal pieces of software that are so complex that it is impossible for any one person to understand the entire system. Windows XP has over 40 *million* lines of code and Red Hat Linux 7.1 has over 30 million [Wheeler 01]. Additionally, with the advent of the World Wide Web, customers are expecting application development to occur in "Internet time" (weeks instead of years), many of which have to be distributed over several to hundreds of computers and systems. Further constraints such as COTS (commercial off-the-shelf) component integration and backwards-compatibility with existing versions make the problem even worse.

Coping with and successfully managing these seemingly impossible requirements have led to the development of countless development processes.[2] However, the plethora of processes and tools available today often leave managers and software practitioners in a quandary as to which process they should choose to develop their software. In this paper, I have chosen to focus on

[1] The waterfall method is where the application development is split into roughly four distinct phases: requirements, design, code, and test. Each phase is completed before proceeding to the next phase. Much like a waterfall, you can move down the progression, but not back up.

[2] Some texts refer to processes and methods synonymously. Here I will use the term process to refer to the series of actions that guide a team through the procedure of developing software.

five of the most popular processes in use today: three "traditional"[3] processes, Rational Unified Process (RUP), Microsoft's Synch and Stabilize (MSS), and Team Software Process (TSP); and two agile processes, Extreme Programming (XP) and Scrum.

1.2 Audience, Purpose, and Goals

In war, battles can be won or lost depending on the weapons that the armies wield. If the weapon matches the situation, the army can be victorious; otherwise the battle will be lost. It is the same with choosing your "weapon" in a software project. Choosing the right development process can help your project succeed, but choosing a process that doesn't match your particular needs can cause your project to ultimately fail.

The goal of this paper is twofold. First and foremost, I hope to provide enough impartial information to educate the novice manager or practitioner as to the tradeoffs and inherent strengths and weaknesses of each of the chosen processes. Secondly, I hope to present both novice and seasoned managers and practitioners a mechanism whereby they can determine which processes are best suited to their particular project, team, and customer. This is done by providing a series of questions designed to help them frame their desired goals within the context of specific processes (namely RUP, MSS, TSP, XP, and Scrum). The questions are backed by relative weights which assign numeric values to a series of attributes, such as team dynamics and project type. These relative weights are supported by empirical evidence presented through case studies and current industrial and academic research.

It must be stressed that the purpose of this paper is not to recommend one process over another blindly. Each process has inherent strengths and weaknesses, and despite the claims made by proponents of each process, there is not a single process that works equally well over every single type of project that exists in industry today. Each process must be evaluated and weighed in the context of a specific project in order to determine the best match.

[3] The term "traditional" process refers to the level of planning and structure involved. I chose this term because in relation to the agile processes, RUP, MSS, and TSP more closely resemble the way development was traditionally done with the waterfall method.

2 Selected Processes Overview

There are literally hundreds of software development processes in use in industry today, ranging from the undocumented, ad-hoc process that is passed on by word of mouth from developer to developer, to the highly structured and rigorous processes that dictate how every aspect of the development life-cycle should be conducted.

This paper focuses on five of the more popular development processes in use today, ranging from the more structured TSP (Team Software Process) to the highly agile Scrum process.

Barry Boehm presents a spectrum of increasing emphasis on plans [Boehm 02], which I have modified slightly by placing the chosen processes on the spectrum (shown in Figure 1). "In this context, the term 'plan' includes documented process procedures that involve tasks and milestone plans, and product development strategies that involve requirements, designs, and architectural plans" [Boehm 02].

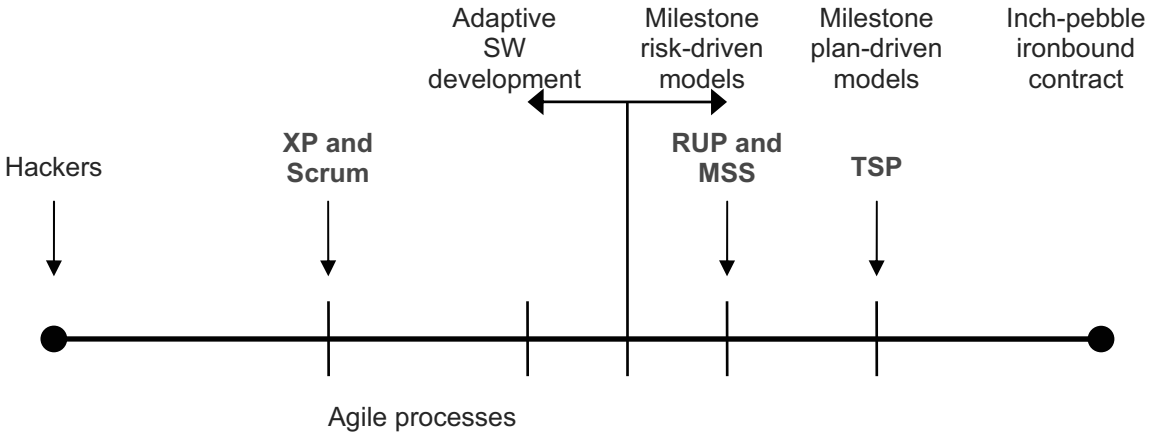

Figure 1: Spectrum of processes discussed in this paper. Unplanned and undisciplined processes are on the extreme left, while micromanaged milestone planning, also known as inch-pebble planning, occupies the extreme right.

A brief background of each method is described in this section, in preparation for the questionnaire in the next section.

2.1 Comparison Criteria

Most processes have many of the same elements, which makes a comparison possible. All processes have people fulfilling specific roles that work together to produce artifacts, which could be the source code, architecture diagrams, requirements documents, and so on. Also, these artifacts usually demarcate milestones in the project, which each process acknowledges but perhaps in different ways.

Each process overview follows the same pattern, which makes it easier to compare the various processes along common criteria. The pattern is a follows:

- **Overview –** a brief description of the process, including any unique characteristics, from the main texts describing the process.
- **Roles –** what specific positions are called for in the process and how are they filled.
- **Artifacts –** what kinds of documents and other artifacts are produced, how often they are produced, and how critical they are to the process.
- **Tools Support –** how many different kinds of tools are available for using the process and what is the cost.

2.2 Rational Unified Process (RUP)

Overview

The Rational Unified Process (RUP, pronounced "rup," not "R. U. P.") provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget [Kruchten 00]. Unlike the other processes discussed in this paper, RUP is a process product, which means that it is developed, maintained, and sold by Rational Software (now owned by IBM). An organization cannot fully use RUP without the accompanying product, although it can use the unified process, which is the precursor and basis of the Rational Unified Process, as described in [Jacobson 99]. Additionally, you do not have to use Rational products for every aspect of the Rational Unified Process (Rational Rose for UML, for example).

RUP embodies six industry best practices within its process framework:

1. Develop software iteratively.
2. Manage requirements.
3. Use component-based architectures.
4. Visually model software.

5. Continuously verify software quality.
6. Control changes to software.

A full description of each of these six practices is outside of the scope of this paper, but can be found in [Kruchten 00], the main source of information about RUP. Point four, however, should be noted. Since Rational is the creator of UML (Unified Modeling Language)[4], it is also a major component of the Rational Unified Process. When using the RUP, you must also use UML: "The Rational Unified Process is a guide to the effective use of the UML for modeling" [Kruchten 00, p. 29].

The famous architectural diagram of the RUP is shown in Figure 2. The process has two dimensions:

- The horizontal dimension represents time and shows the lifecycle aspects of the process as it unfolds.
- The vertical dimension represents core process disciplines (or workflows), which logically group software engineering activities by their nature [Kruchten 01].

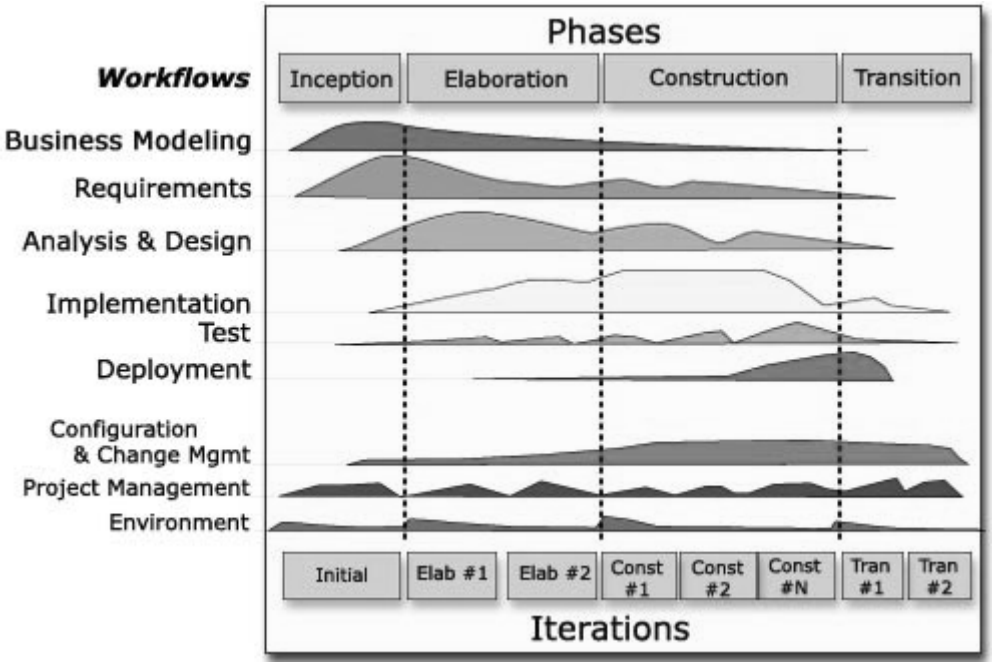

Figure 2: Two Dimensions of RUP

[4] The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system [Kruchten 00, p. 28].

Roles

RUP defines over 30 roles in its process. At first glance, this may seem an inordinate amount and may be one of the reasons why it has a stigma associated with it as being "heavyweight." It is important to understand how RUP defines a role, however, before judging whether the Rational Unified Process is too monolithic to use for your particular project.

Roles were formerly called *workers* in previous versions of the RUP. A *worker* defines the behavior and responsibilities of an individual or a group of individuals working together as a team. The responsibilities of each worker are usually expressed in relation to certain *artifacts* that the worker creates, modifies, or controls. It's helpful to think of a worker as a "hat" that an individual can wear during the project. One person may wear many hats. This distinction is important because it is natural to think of a worker as the individual or the team, but in the Rational Unified Process the term *worker* refers to the roles that define how the individuals should do the work [Kruchten 00].

Artifacts

RUP prefers to call deliverables *artifacts* rather than documentation. Artifacts are the tangible products of the project: the things the project produces or uses while working toward the final product. Typically, artifacts are *not* documents. RUP claims that it "discourages the systematic production of paper documents" [Kruchten 00], however, there are over 60 artifacts defined in the Rational Unified Process. It should be noted that not all of these artifacts will be produced in any given project. Also, RUP is designed to be tailored to an organization's particular environment and the specific project that is under development.

Tools Support

The good news in RUP is that there are many tools available, mostly from Rational, to aid in using the Rational Unified Process in your organization. The bad news is that RUP requires a lot of tool support because of all of the artifacts. As a bare minimum, you need the RUP tool itself, a UML tool, a word processor, templates for the paper-based documentation, a version control system, and a requirements change management tool. Most of these tools are available from vendors other than Rational. However, Rational's tools are integrated together to form a coherent whole.

2.3 Microsoft's Synch-and-Stabilize Process (MSS)

Overview

Any discussion about software process would be incomplete without at least mentioning the process that the largest and most successful software company in the world uses. Unfortunately, there is not a lot of printed material about the details of Microsoft's internal development process. The main work on this subject is the book *Microsoft Secrets* [Cusumano 95], written by two independent researchers, Michael Cusumano and Richard Selby. Although the book was written in 1995, Microsoft's basic process has not changed much since that time. It should also be noted that Microsoft has another process which they tout, called the Microsoft Solutions Framework (MSF). This process is used by Microsoft Consulting Services to help other software organizations in their development process. While encompassing many of the internal processes used within Microsoft, it does not completely describe the synch-and-stabilize approach used to develop most of Microsoft's products.

The term "synch and stabilize" (hereafter abbreviated as MSS) was coined by Cusumano and Selby in their book to describe Microsoft's milestone-driven development process. To quote them directly:

> *We have labeled Microsoft's style of product development the* synch-and-stabilize *approach. The essence is simple: continually* synchronize *what people are doing as individuals and as members of different teams, and periodically* stabilize *the product in increments—in other words, as the project proceeds, rather than once at the end. When team members build components that are interdependent but difficult to define accurately in the early stages of the development cycle, the team must find a way to structure and coordinate what the individual members do while allowing them enough flexibility to change the product's details in stages. This is useful to do as developers test the product with customers and refine their designs* during the development process.[5]

There are three main phases in MSS: planning, development, and stabilization.[6] These phases are done serially, but within each phase development can happen in parallel amongst separate feature teams.

Planning Phase

The planning phase usually takes 3-12 months and encompasses defining the product vision, specification, and schedule. The three main steps are:

---

[5] Cusumano 95, p. 14.

[6] The phases are summarized from a diagram in Cusumano 95, p. 194.

- **Vision Statement:** Product and program management use extensive customer input to identify and prioritize product features.
- **Specification Document:** Based on the vision statement, program management and the development group define feature functionality, architectural issues, and component interdependencies.
- **Schedule and Feature Team Formation:** Based on the specification document, program management coordinates the schedule and arranges feature teams that each contain approximately 1 program manager, 3-8 developers, and 3-8 testers (who work in parallel 1:1 with developers).

Development Phase

The development phase is where the coding takes place. Feature development occurs over 3 or 4 sequential subprojects that each result in a milestone release. Program managers coordinate the evolution of the specification. Developers design, code, and debug. Testers pair up with developers for continuous testing.

- **Subproject I:** The first third of the features are developed, which are the most critical features and/or shared components.
- **Subproject II:** The second third of the features are developed.
- **Subproject III:** The final third of the features are developed, which are the least critical features.

Stabilization Phase

In this phase, comprehensive internal and external testing, final product stabilization, and shipping the product are performed. Program managers coordinate OEMs (Original Equipment Manufacturers) and ISVs (Independent Software Vendors) and monitor customer feedback. Developers perform final debugging and code stabilization. Testers recreate and isolate errors.

- **Internal Testing:** Thorough testing of the complete product within the company.
- **External Testing:** Thorough testing of the complete product outside the company by "beta" sites such as OEMs, ISVs, and end-users.
- **Release Preparation:** Prepare the final release of "golden master" diskettes and documentation for manufacturing.

Roles

Although there are several types of supporting roles in MSS, there is a triumvirate of three major roles that make up a core team: program manager, developer, and tester. Each role is considered vital and equal to each other in importance. One cannot exist without the other.

Program Manager (PM)

The program manager (PM) serves as a critical link between the developers and the marketing department. He or she ultimately oversees the development process to make sure that it aligns with customer needs and requirements. It is important to point out that the program manager "is a leader, facilitator, and coordinator, but is *not* the boss."[7] Also, the PM is not the designer. Although the program manager can have a key part in the design process, as Bill Gates says, "development is still … in the strong position, let's face it. If they have an idea, they get to write the code."[8] The key areas of responsibility for program managers are:

- the product's vision
- the written product specification
- the product schedule
- the product development process
- all implementation trade-offs
- coordination of the product development groups

Developer (Software Design Engineer, or SDE)

Program managers generally focus on the vision for the overall product and what types of features make up this vision. Developers, in contrast, define the vision and create the details of individual product features. Developers "write the code, they implement the features, they know the code base. Their job is to ship products by writing code."[9] The developer's main responsibilities are summarized below:

- determine the vision for new features
- design the features
- allocate project resources
- build the features
- test the features
- prepare the product for shipping

---

[7] Ibid., p. 77.

[8] Ibid., p. 79.

[9] Cusumano 95, p. 82-83.

Tester (Software Test Engineer, or STE)

Many of the other processes presented in this paper treat development and testing as either a single activity or the role is filled by the same person. Microsoft, however, treats testing as a separate discipline filled by a different person. Each developer has a testing "buddy" and the two of them work very closely together. Microsoft summarizes why it treats testing as a separate discipline in three ways: (1) Developers do not produce perfect code, and program managers do not produce perfect specifications; (2) it is necessary to have someone detached from the spec and the code provide an unbiased perspective on their quality; and (3) it is much less expensive and easier for developers—as well as much better for product reliability and customer satisfaction—to find and fix bugs as early as possible in the development process, when pieces of code are less intertwined.[10]

Other Roles

MSS has three more well-defined functions that also overlap with the other roles. *Product managers* are marketing specialists; *customer support engineers* provide technical support to users and analyze customer feedback; and *user education staff* prepares manuals and help documentation.

Artifacts

MSS has two main pieces of documentation that are essential to any project: the vision document and the specification document. These two pieces of documentation are written in the planning phase and define the scope and vision of the project. Other pieces of documentation are schedules from the program manager, code from the developers, and test plans and automated test cases from the testing group.

Tools Support

Although there is a wealth of internal tools support within Microsoft, unfortunately there are not a lot of commercially available tools to support using the synch-and-stabilize process "out of the box." There are, of course, tools to support documentation and planning that the PM must do, development tools for the developer, and testing tools that support the tester, but there is not one comprehensive "synch-and-stabilize product" or suite of products (such as Rational has for their Unified Process).

---

[10] Ibid., pp. 85-86.

2.4 Team Software Process (TSP)

Overview

The Team Software Process (TSP)[11] was created by Watts Humphrey of Carnegie Mellon's SEI (Software Engineering Institute). It "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" [Humphrey 00]. Virtually every aspect of the development process is codified in a series of step-by-step scripts. Since it builds upon the Personal Software Process (PSP), even the process of writing code is documented in a script. Each script contains a set of entry criteria, actions, and exit criteria that determine when the actions on the script have been completed. The development process is iterative in nature (each iteration is called a cycle in TSP), with each successive iteration building upon its predecessors until finally you have a completed project.

There are two different flavors of TSP: one targeted towards an academic environment (TSPi) and another targeted towards industry use. Although there are slight variances, the process is similar enough that I will use TSPi as the basis for discussion in this paper. The main source of information for TSPi is the book *Introduction to the Team Software Process* by Watts Humphrey.

Roles

There are five main roles in TSP: team leader, development manager, planning manager, quality/process manager, and the support manager. Each person, besides filling one of the above roles, acts as a developer and writes code. This is the default behavior for the academic version, TSPi, but the roles can be separated in the industry version of TSP.

Team Leader

The team leader knows what needs to be done, and is willing to insist that the team members do their work as they know they should. He or she must settle disputes and must maintain the team's energy and pace while also taking advantage of everyone's creative ideas and abilities.[12]

The eight principle responsibilities of the team leader are as follows:

1. Motivate the team members to perform their tasks.
2. Hold a team meeting every week.

---

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

[12] Humphrey 00, p. 217.

3. Report team status and progress to superiors every week.
4. Lead the team in allocating tasks among the team members.
5. Act a facilitator and timekeeper in all the team meetings.
6. Maintain the project notebook.
7. Lead the team in producing the development cycle report.
8. Act as a development engineer.[13]

Development Manager

The development manager's specific goal is to guide the team in producing a superior product. The measure of success in achieving this goal is that the team produces a useful and fully documented product that meets the basic requirements of the need statement.[14]

The 11 principle responsibilities of the development manager are as follows:

1. Lead the team in producing the development strategy.
2. Lead the team in producing the preliminary size and time estimates for the products to be produced.
3. Lead the development of the software requirements specification.
4. Lead the team in producing the high-level design.
5. Lead the team in producing the software design specification.
6. Lead the team in implementing the product.
7. Lead the development of the build, integration, and system test plans.
8. Lead the team in developing the test materials and running the tests.
9. Lead the team in producing the product's user documentation.
10. Participate in producing the development cycle report.
11. Act as a product developer.[15]

Planning Manager

The planning manager's principal goal is to help and support the team in producing a complete, precise, and accurate project plan. A second planning manager goal is to accurately track team progress and produce a weekly project status report that the team leader uses to report status to his or her superiors.[16]

The six principal responsibilities of the planning manager are as follows:

---

[13] Humphrey 00, p. 217.

[14]Humphrey 00, p. 232.

[15] Humphrey 00, p. 233.

[16] Humphrey 00, p. 248.

1. Lead the team in producing the task plan for the next development cycle.
2. Lead the team in producing the schedule for the next development cycle.
3. Lead the team in producing the balanced team-development plan.
4. Track the team's progress against the plan.
5. Participate in producing the development cycle report.
6. Act as a product developer.[17]

Quality/Process Manager

The quality/process manager's principal goal is to help the team members record and use their TSP data, to guide the team in faithfully using the TSP to produce a quality product, and to perform effectively as the team's inspection moderator and meeting recorder.[18]

The nine principal responsibilities of the quality/process manager are as follows:

1. Lead the team in producing and tracking the quality plan.
2. Alert the team, the team leader, and any superiors to quality problems.
3. Lead the team in defining and documenting its processes and in maintaining the process improvement process.
4. Establish and maintain the team's development standards and the system glossary.
5. Review and approve all products before submission to the CCB (change control board).
6. Act as the team's inspection moderator.
7. Act as recorder in all the team's meetings.
8. Participate in producing the development cycle report.
9. Act as a product developer.[19]

Support Manager

The support manager's principal goals are to ensure that the team has suitable tools and methods to support its work, to make sure that there are no unauthorized changes to baselined products, to record and track all risks and issues, and to help the team meet its reuse goals.[20]

---

[17] Humphrey 00, p. 249.

[18] Humphrey 00, p. 264.

[19] Humphrey 00, p. 265.

[20] Humphrey 00, p. 276.

The eight principal responsibilities of the support manager are as follows:

1. Lead the team in determining its support needs and in obtaining needed tools and facilities.
2. Chair the configuration control board and manage the change control system.
3. Manage the configuration management system.
4. Establish and maintain the system glossary.
5. Handle the team's issue- and risk-tracking.
6. Act as the team's reuse advocate.
7. Participate in producing the development cycle report.
8. Act as a development engineer.[21]

Artifacts

The majority of the TSP's artifacts are paper-based documentation in the form of plans, scripts, documents, and reports. Some of the notable artifacts are the SRS (Software Requirements Specification), SDS (Software Design Specification), test plan, and planning document. "Out of the box," there are over 20 forms, scripts, and written documentation that are required to be filled out. This makes TSP the most document-intensive process examined in this paper. This is not a bad thing, however, as the comprehensive scripts and forms allow novice or new teams to quickly become productive.

Since the process is highly structured and documented, it facilitates rapid ramp-up times for new teams and team members. This is especially evident in team projects where every person on the team might be unfamiliar with each other and each individual team members' strengths and weaknesses. Much of the time spent trying to set up roles and processes within the team are handled by the scripts. Furthermore, each role is defined so well that almost any person can fill the role, without prior experience or expertise. A team lead does not necessarily have to be the strongest leader, for example, or the QA person does not have to have formal training in testing and verification.

There seems to be a point at which the highly structured script-based process can impose a lot of overhead to the development work. For certain project domains and applications, this overhead is necessary and indispensable, such as mission- or safety-critical systems where human lives are at stake. This overhead can be a weakness to the process if the problem domain does not call for it. Although proponents of the TSP would argue that the process can be tailored to the individual needs of the team if the team finds that it is too heavyweight,

---

[21] Humphrey 00, p. 277.

ironically it must be done in a highly structured way by filling out a PIP (Process Improvement Proposal).

Tools Support

Although there are some rudimentary tools to support the TSP, most of the tools come in the form of comprehensive forms and scripts (as described above). Most of the forms are available as Excel spreadsheets to facilitate easy data entry.

2.5 Extreme Programming (XP)

Overview

Extreme Programming (XP) is perhaps one of the most talked about processes as of late. In many people's minds it epitomizes the Agile methodology. The general principle behind XP is that software is nothing without the code. Anything detracting from writing code had better have a good reason for doing so. This includes detailed plans and designs, which is one reason why XP has drawn some criticism from opponents.

There are 12 main tenets of XP, summarized below.[22]

- *The Planning Game*—Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan.

- *Small releases*—Put a simple system into production quickly, and then release new versions on a very short cycle.

- *Metaphor*—Guide all development with a simple shared story of how the whole system works.

- *Simple design*—The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.

- *Testing*—Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating that features are finished.

- *Refactoring*—Programmers restructure the system without changing its behavior to remove duplication, improve communications, simplify, or

[22] Beck 00, p. 54.

add flexibility.

- *Pair programming*—All production code is written with two programmers at one machine.

- *Collective ownership*—Anyone can change any code anywhere in the system at any time.

- *Continuous integration*—Integrate and build the system many times a day, every time a task is completed.

- *40-hour week*—Work no more than 40 hours a week as a rule. Never work overtime a second week in a row.

- *On-site customer*—Include a real, live user on the team, available full-time to answer questions.

- *Coding standards*—Programmers write all code in accordance with rules emphasizing communication through the code.

Roles

XP defines 7 different roles, all of which could be filled by the same person. The only exception is customer, which should optimally be a person not associated with any other role on the team. Five of the seven roles are primary roles, with the other two being secondary roles and are not always needed on an XP project.

Programmer

"The programmer is the heart of XP. Actually, if programmers could always make decisions that carefully balanced short-term and long-term priorities, there would be no need for any other technical people on the project besides programmers."[23]

Programmers are responsible for writing the code in pairs, one who "drives," and the other who ensures that the code is defect-free. In this way, reviews are built into the coding process. Programmers also must use the "test-first" mentality by writing unit tests that test the feature to be implemented before it is coded.

---

[23] Beck 00, p. 141.

Customer

"The customer is the other half of the essential duality of extreme programming. The programmer knows how to program. The customer knows what to program."[24]

Customers are responsible for determining what features the system must have and which ones can be postponed until the next version. Additionally, the customer must be adept at writing stories, which are the basis behind XP's planning game. The customer must also write functional acceptance tests, with the goal being able to say, "Well, if these run, then I'm confident the system will run."[25]

Tester

Unlike Microsoft's synch-and-stabilize process, XP does not have separate testers. The programmers are also the testers. However, there is a role that can be filled by one of the programmers in making sure the customer's acceptance tests are included in the test suites. As Kent Beck says, "An XP tester is not a separate person, dedicated to breaking the system and humiliating the programmers. However, someone has to run all tests regularly (if you can't run your unit and functional tests together), broadcast test results, and to make sure that the testing tools run well."[26]

Tracker

The tracker is responsible for keeping a project history. He or she keeps a log of functional test scores, defects reported, who accepted responsibility for each, and what test cases were added on each defect's behalf. Additionally, he or she keeps track of approximately how much time each programmer spent on his or her task. In this way, when programmers are estimating how much time it will take to complete a story, the tracker will be able to say, "Two thirds of our estimates last time were at least 50% too high."[27]

Coach

The coach is responsible for the process as a whole. He or she notices when people are deviating from the team's process and brings this to the team's

---

[24] Beck 00, p. 143.

[25] Beck 00, p. 144.

[26] Beck 00, p. 144.

[27] Beck 00, p. 143.

attention. Although everyone on the team is responsible for understanding their application of XP to some extent, the coach is responsible for understanding it much more deeply—what alternative practices might help the current set of problems; how other teams are using XP; what the ideas behind XP are; and how they relate to the current situation.[28]

Consultant

Since everybody on an XP team shares ownership of the code, XP projects don't spawn a lot of specialists. Usually this is a strength, but occasionally the project will call for a specialist in a particular area. When it does, the team needs a consultant, usually brought in from the outside.[29]

Big Boss

The role of the big boss is to provide an environment that is conducive to XP development. This means that if the boss is not familiar with XP, that he should trust his team and developers that they know what they're doing even if it appears unconventional.

Artifacts

The primary artifact in XP is the code. Everything in the process is tailored to get out as much code as possible in a given amount of time. However, there are pieces of non-code artifacts that are often overlooked. Story cards, for example, are a critical piece of the XP process and can encompass a huge amount of printed pages if they were to be typed out. Since they are handwritten for the most part, they are often overlooked as part of the process's documentation. There are also code standard documents, tracking and time estimation reports, and an overall plan document. Of course, the XP mindset says that the documents are as thin as sparse as possible. Therefore, compared to other processes presented in this paper, the two Agile processes (XP and Scrum) definitely have less non-code artifacts.

Tools Support

The main tool support needed for XP is an automated unit testing tool. There are several quality tools available on the market for a variety of platforms and languages. Also, there are several types of planning and defect tracking tools commercially available, so there is really no practical limitation from lack of tool support in using XP as a development process.

---

[28] Beck 00, p. 145.

[29] Beck 00, p. 146-147.

2.6 Scrum

Overview

Scrum[30] is not so much a development process as a management process. Scrum can be wrapped around an existing development process, and has been used quite successfully in combination with XP. By itself, Scrum cannot produce software; it must be combined with another development process. It can simply organize how the software is developed.

Scrum's goal is to deliver as much quality software as possible within a series (3-8) of short time-boxes (fixed time intervals) called *Sprints* that typically last about a month. Each stage in the development cycle (Requirements, Analysis, Design, Evolution, and Delivery) is now mapped to a Sprint or series of Sprints. The traditional software development stages are retained for convenience primarily for tracking milestones. So, for example, the Requirements stage may use one Sprint, including the delivery of a prototype. The Analysis and Design stages may take one Sprint each, while the Evolution stage may take anywhere from 3 to 5 Sprints. The typical Scrum process flow is shown in Figure 3.[31]

Each Sprint operates on a number of work items called a *Backlog*. As a rule, no more items are externally added into the Backlog within a Sprint. Internal items resulting from the original pre-allocated Backlog can be added to it. The goal of a Sprint is to complete as much quality software as possible, but typically less software is delivered in practice. The end result is that there are non-perfect named stable bases delivered every Sprint.

---

[30] Scrum is sometimes seen in all caps (SCRUM) and just the initial letter capitalized (Scrum). The name Scrum refers to the mechanism used in rugby for getting an out-of-play ball back into play.

[31] The following text is taken almost verbatim from [Beedle].

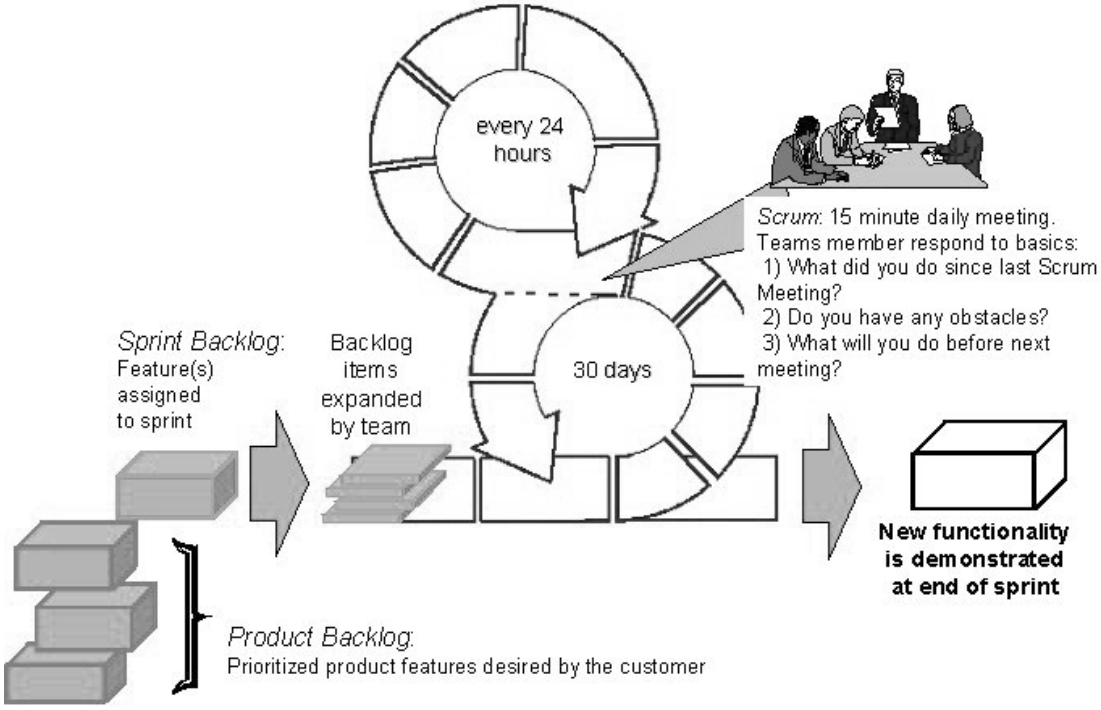

Figure 3: Scrum Process Flow[32]

During a Sprint, *Scrum Meetings* are held daily to determine on:

1. What items were completed since the last Scrum Meeting.
2. What issues or blocks have been found that need to be resolved.
3. What new assignments make sense for the team to complete until the next Scrum Meeting.

Roles

There is really only one role in Scrum: the *Scrum Master*. The Scrum Master has a number of primary responsibilities, mostly revolving around the master backlog list which he or she controls. He or she also acts as a *firewall*, shielding the development team from upper management or the customer. For example, nobody is allowed to add to the backlog during a Sprint cycle that has already started. If upper management or the customer wants to add to it they can, but they must do so at the expense of cutting the Sprint cycle short and losing much of the work that had already been accomplished. This shielding from outside requirements changing for the duration of the Sprint allows the development team to focus on the current tasks at hand.

---

[32] Diagram taken from the Scrum web site, http://www.controlchaos.com.

Probably the most important responsibility of the Scrum Master is to conduct the daily 15 minute Scrum Meetings, in which he or she tracks progress and identifies any impediments to progress.

Artifacts

Only one artifact, the master backlog, is presented in Scrum. The backlog can be as simple as a list in an Excel spreadsheet and requires very minimal management overhead.

Tools Support

Since there is only a requirement to have a backlog, and not a prescribed method to produce it, any commercially available spreadsheet program is sufficient for implementing Scrum.

2.7 Process Summary

What is common amongst all of the processes presented is the concept of iterations. No longer is software developed in large chunks, as proposed by the waterfall method. Instead, each iteration builds upon its predecessors building the system one small piece at a time. Risk is handled by a series of milestones, checkpoints, Sprint meetings, etc. Each process at its core deals with this important question on how to manage risks associated with software development.

3 Choose Your Weapon

Having read through the brief overview of the processes discussed in this paper, you probably have a rough idea of which processes may best apply to your specific project. The point of this section is to present a series of questions designed to help guide and direct your thoughts even further so that you may ultimately come up with one or two candidate development processes that will best suit your project.

The questions have been divided into four main categories: team and product size, developers and organization, product, and requirements. Each question has relative weights assigned to each multiple choice answer for each of the five processes discussed in this paper. The weights range from 1 to 3, where 1 means an inherent weakness, 2 is no weakness or strength, and 3 is an inherent strength. For example, Scrum would score a 3 in a question regarding the amount of overhead associated with keeping artifacts up to date, but TSP may score a 1. A running tally of these relative weights can be examined at the conclusion of the questionnaire to provide you with a ranked list of process suitability. It is important to note that the order of presentation does not imply the order of importance. A brief discussion of why the weights are assigned the way they are follows each question.

3.1 Team and Product Size

Team Size

How many developers and testers are involved in a single team within the project (on average)?

a) *Less than 10*
b) *10-20*
c) *More than 20*

Scrum works optimally with teams of 7 or less ("There is plenty of data to show that team sizes over 7 result in significantly lower productivity. Any team over 7 in size should be split up into multiple SCRUMs" [Sutherland 03]).[33] Team size

[33] "I described how a few teams in a 500 person development group generated production code at five times the industry average, while most of the teams who executed SCRUM well, only doubled productivity over industry average. One of the problems in the large organization is

in XP is limited to 10 people ([Beck 00] says that you probably couldn't run an XP project with 20 programmers, but that 10 is "definitely doable.") TSP defines five roles which should optimally be filled by different people (team lead, development manager, planning manager, quality/process manager, and support manager). XP defines 4 required roles (programmer, coach, tester, and tracker), but a programmer can fill any of the other 3 roles. RUP defines a plethora of roles, but each can be filled by the same person (there are about 30 in Kruchten's book). However, "for each worker, a set of expected skills must be provided by the individual who is designated as the worker" [Kruchten 00, p. 37]. Therefore, a small team working with RUP may have to tailor the process significantly to accommodate their size.

Matrix

The matrices presented at the end of each question give you the ability to see how one process ranks in comparison to another. Use the tally sheet in Appendix A to keep track of your answers.

	RUP	MSS	TSP	XP	Scrum
a	1	1	3	3	3
b	2	2	1	1	1
c	3	3	1	1	1

a) TSP, XP, and Scrum are preferred, since they are designed for small teams "out of the box." RUP and will not work well right out of the box and would need to be modified for small teams. MSS requires a tester matched to a developer, so small teams do not work quite as well.

b) RUP and MSS work equally well here. TSP, XP, and Scrum would have to be seriously modified to let them work with large teams.

c) RUP and MSS excel on large teams compared to the other three processes. TSP, XP, and Scrum could possibly work, but there is not a lot of industry data to support this claim. In fact, there are several articles which claim that agile processes do not ramp up that well to large teams without serious modification.

---

that it was culturally prone to a team size of about 15 people and there was a lot of internal resistance to reducing team size. I now think that this may be the primary reason only a few teams moved into hyperproductive mode. The hyperproductive teams would always split into subgroups of 7 or less, while the poorer performing teams insisted on working as a group of 15."

Total Developers

How many developers are involved in the entire project?

a) *Less than 40*

b) *41-100*

c) *Hundreds*

RUP and MSS can handle hundreds of people over many different teams, which has been proven with an industry track record. Microsoft continually produces products that span multiple teams with sometimes hundreds of developers and testers. There are numerous case studies using RUP in large projects with hundreds of developers, the Volvo case study for one[34]. TSP can be used on large multi-teams but "additional process extensions are required for larger teams" [SEI-TSP 03]. Agile methods tend to break down past 20-40 people, since face-to-face communications is critical [Lindvall 02].

Matrix

	RUP	MSS	TSP	XP	Scrum
a	2	2	2	2	2
b	3	3	1	1	2
c	3	3	1	1	2

a) Any process could work.

b) XP and TSP start to break down at this size. RUP and MSS work well. Scrum can work, but modifications necessary.

c) RUP and MSS are very effective at this size. XP and TSP effectively don't work. Scrum has been used on an 800-person project [Schwaber 02], but had to be modified to accommodate "scrums of scrums."

Product Size and Complexity

What is the size of your product in terms of lines of code and complexity?

a) *Hundreds to a few thousand (small to medium independent programs such as Notepad or Adobe Acrobat Reader)*

b) *Hundreds of thousands to a few million (large business applications such as Microsoft Word, Adobe Photoshop, or Rational Rose)*

---

[34] The Volvo IT group switched to RUP successfully, spanning hundreds of developers working over several different projects.

c) *Millions (huge systems such as an operating system or the Space Shuttle navigation system)*

XP is not suitable for projects with over 40,000 LOC (roughly) (that number comes from using a maximum of 10 people over a fifteen month cycle programming in Java) [Smith 01]. RUP and MSS can scale to any sized project, but favor mid to large size projects. RUP and TSP provide specific guidelines as to what documentation to produce to allow large development. MSS is a little vaguer as to specific documents, but it certainly is used in large project development.

Matrix

	RUP	MSS	TSP	XP	Scrum
a	1	1	1	3	3
b	2	2	2	2	2
c	3	3	2	1	1

a) XP and Scrum have the least amount of overhead with projects of this size compared to RUP and MSF.

b) Scrum, RUP, MSS, or TSP all work well, but XP is still possible.

c) TSP and RUP may be preferred here as they provide specific guidelines for producing artifacts (although TSP is not proven on projects of this size). Although MSS does not provide as much specific guidance, it has been successfully used in projects of this magnitude. Scrum and XP could work, but they have not been used on projects of this magnitude (at least they have not been documented on projects of this magnitude in articles).

3.2 Developers and Organization

Competent and Experienced Developers

What percentage of your developers are "competent and experienced," where "competent" means: 1) Possess real-world experience in the technology domain; 2) Have built similar systems in the past; 3) Possess good people and communication skills?[35]

a) Less than 10%
b) 10%-25%
c) 25%-33%

[35] Lindvall 02, p. 202.

d) 33%-100%

It is generally acknowledged that Agile methodologies require a good percentage of competent people. Lindvall, et al. suggests that at least 25%-33% of developers should be experienced for an Agile process to be successful. Agile processes do not require everybody to be highly capable, however. The main difference between XP and TSP, RUP, and MSS is that XP "derives much of [its] agility by relying on the tacit knowledge embodied in the team, rather than writing the knowledge down in plans" [Boehm 02]. TSP is an excellent choice for teams of novice developers because most of the management and development is codified in a series of scripts.

Matrix

	RUP	MSS	TSP	XP	Scrum
a	1	1	3	1	1
b	2	2	2	1	1
c	2	2	2	2	2
d	3	3	1	3	3

a) TSP is the clear winner. RUP and MSS require too much learning curve for inexperienced developers. XP and Scrum need at least 25%-33% of "competent and experienced" developers.

b) TSP, RUP, and MSS provide enough guidance for the novice developer. Scrum and XP would not work well here. XP could work if pair programming teams were rotated to have the experienced people program with the novice people, but it would not be as effective.

c) TSP, RUP, and MSS are about equal. XP and Scrum will work, but not as effectively as the other three, especially on larger projects where this "tacit knowledge" is unable to be contained in a single mind.

d) XP and Scrum thrive in this environment. TSP may be too simplistic, especially if the developers are hackers at heart.[36] RUP and MSS would also likely succeed with a larger percentage of experience developers.

[36] See [Himanen 01] for a complete and thorough discussion on what constitutes a "hacker."

Level of Hacker Sentiment

Which of the following phrases accurately describes the predominant sentiment on your team?

a) *"We've talked about the system long enough, let's just code the thing. We can always rewrite stuff that turns out to be wrong faster than if we had spent a lot of time trying to get it perfect the first time."*

b) *"We should plan and design enough of the system to feel confident that we haven't overlooked anything critical, but we should not try to get everything perfect before starting to code."*

c) *"We better make sure that we have thought through all of the possible scenarios of our system before we start writing code. Without a formal plan and design, we may make serious mistakes that will cost us later down the road."*

XP and Scrum (and all Agile processes) place an emphasis on people over process: "Individuals and interactions over processes and tools" [Manifesto]. People who are hackers at heart[37] prefer coding over documentation and project management. XP and Scrum are perfectly suited to these types of people. MSS and RUP tend to take a more structured approach to development, but still allow a lot of flexibility. Cusumano and Selby state, "We believe no PC software company has done a better job of keeping some basic elements of the hacker culture while adding just enough structure to build today's and probably tomorrow's PC software products.… Microsoft still encourages some teams to experiment and make lots of changes without much upfront planning."[38] TSP is the most structured, but also places the most emphasis on up-front planning and design before any code is written.

Matrix

	RUP	MSS	TSP	XP	Scrum
a	1	3	1	3	3
b	2	2	1	2	2
c	3	2	3	1	1

a) MSS, XP, and Scrum. RUP and TSP require more up-front planning and design.

---

[37] See [Himanen 01] for a complete and thorough discussion on what constitutes a "hacker."

[38] Cusumano 95, p. 16.

b) RUP and MSS are favored, but Scrum and XP will work too. TSP stresses that you shouldn't start coding until everything has been thought through.

c) This sentiment typifies TSP, but RUP and MSS accommodate this sentiment just as well. XP and Scrum were invented to avoid this sentiment.

Management Style

What kind of management style best suits your project?

a) Macro-management: Only a few guidelines are outlined by the manager and developers have to fill in the gaps on their own.

b) Median-management: Most of the important aspects of the project are documented by the manger, but developers still have a lot of leeway in designing and implementing the system.

c) Micro-management: Almost all of the aspects of the project are documented and developers don't have much control over what they write.

There is a general stereotype that RUP is "heavyweight" and XP is "lightweight." As Smith's article (RUP vs. XP) points out, there really is no difference between the two as far as what is produced as artifacts. However, while this differentiation between RUP and XP seems to be mythical, XP "feels" lightweight to programmers in comparison to RUP. Paraphrasing the words of an experienced developer, Alistair Cockburn says, "A small, rigorous methodology may *look* the same as an agile methodology, but it won't *feel* the same" [Cockburn 01]. XP gives the programmer role a lot of control over his own development, whereas RUP tends to put non-programming roles in control (use case designer or architect, for example). MSS, embodied in Microsoft's internal process, seems to balance the tension between unguided, self-motivated development and micro-managed development. TSP sways to the micro-managed side, but in an interesting way. The roles are so well defined that there is little room for variance. However, the person in the role is free to do things differently; it just must be done in a structured way. In a sense, then, the person is micromanaging himself.

Matrix

	RUP	MSS	TSP	XP	Scrum
a	2	3	1	3	3
b	3	3	2	2	2
c	2	2	3	1	1

a) MSS, XP, and Scrum. RUP could work, but not as well. TSP doesn't work in this kind of environment.

b) MSS and RUP come configured this way "out of the box." Scrum and XP from the light-weight side, and TSP from the heavy-weight side can be tailored to become middle-weight.

c) TSP. MSS and RUP could be configured this way. Scrum and XP by their nature do not allow micro-managing.

Organization-Wide Processes

How important is it to your organization to develop organization-wide processes, rather than project-specific processes?

a) Critical. The company wants to protect against staff turnover by retaining the knowledge acquired on each project.

b) Somewhat important. Although there is occasional staff turnover, there are enough people left to retain the collective knowledge in their minds.

c) Not important at all. Your organization is a small company where a high staff turnover would not matter because the company would most likely go out of business anyway.

RUP *has* to be configured as a required step in RUP itself, which promotes tailoring the process (and documenting it) at the front-end of the project lifecycle. XP is less formal in its administration, and therefore spreads out the learning curve and adoption over the life of the project. "An organization will also, more than likely, tailor RUP for organization-wide application on particular types and sizes of projects, and will use the results in several projects… XP does not obviously motivate the capture of 'corporate memory,' leaving an adopting organization (if it does not save its process experience) vulnerable to staff turnover" [Smith 01, p. 20]. MSS is more similar to RUP than XP in this regard. In fact, it came about because in the early days of Microsoft, key players leaving the team would take a large chunk of collective knowledge with them. TSP is very similar to RUP.

Matrix

	RUP	MSS	TSP	XP	Scrum
a	3	3	3	1	1
b	2	2	2	2	2
c	2	2	2	3	3

a) RUP, MSS, or TSP. Scrum and XP can work too, but steps have to be taken to build in process documentation into the Scrum sprints and XP cycles.

b) Any process could work here.

c) XP and Scrum require no changes to their process. RUP, MSS, and TSP can still produce their process documentation with no adverse effects other than higher overhead where it may not be required.

New Process Adoption

What level of confidence do you require before adopting a new process for your project?

a) The process must be "tried-and-true," having been around for several years and used in a variety of market segments.

b) The process is fairly mature, but has not been used in a wide variety of market segments.

RUP, MSS, and TSP have been around longer than XP and Scrum. RUP has the biggest variance of market segments, being used in the automotive, medical, financial, educational, and government markets. MSS has been used predominately in the business and consumer markets. TSP has been used mostly in government and educational settings. MSS, TSP, XP, and Scrum have about the same level of market variance. RUP and XP have the largest market share (in terms of number of different companies using them) in comparison to Scrum, MSS, and TSP.

Matrix

	RUP	MSS	TSP	XP	Scrum
a	3	3	1	2	1
b	2	2	2	2	2

a) RUP and MSS first, then XP, then Scrum and TSP.

b) This describes all of the processes but RUP, although RUP can be used here too.

3.3 Product

Type of Product

What type of product do you have?

a) *Life-critical, such as an air traffic control system or patient monitoring system*

b) *Non life-critical, but mission-critical system, such as a stock exchange or banking application*

c) *An embedded system that is neither life- nor mission-critical*

d) *An application that is neither life- nor mission-critical*

XP and Scrum do not have enough industry experience in mission- or life-critical systems. Scott Ambler, the originator of agile modeling says, "I would also be leery of applying agile modeling to develop life-critical systems, such as an air traffic control system or patient monitoring system, simply because I don't work on such projects and have no insights into how well AM will work on them" [Ambler 01]. Theoretically, "projects developed with XP can adhere to strict (or safety) requirements" [Lindvall 02], but there isn't enough experience yet in this arena.

Matrix

	RUP	MSS	TSP	XP	Scrum
a	3	2	3	1	1
b	2	2	2	2	2
c	3	3	2	2	2
d	2	2	2	2	2

a) TSP because of its strict documentation and review process, although it has not been used in this type of application before (at least it is not documented). RUP can certainly accommodate life-critical systems and has been used in the medical and automotive industries. MSS could be tailored for life-critical systems, but anybody who has used a Microsoft product may not feel completely comfortable entrusting his or her life to it.[39]

b) Any process could work just as well.

c) RUP or MSS. Scrum, XP, and TSP could work, but not enough industry experience to back the claims up.

d) All of the processes have been successfully used for these types of applications.

[39] This is not an attack against Microsoft. In fact, it should be noted that the author now works at Microsoft.

3.4 Requirements

Requirements Stability

How stable are the requirements from the outset of the project?

a) *Emergent and rapidly changing*

b) *Knowable early and largely stable*

It is rare that a project has stable requirements over the lifetime of the project. Some projects, however, really do have fairly stable requirements, such as some government projects. All of the processes seek to enable the practitioners to respond to requirements changes over the lifetime of the project by introducing incremental development, but some do it better than others. The Agile processes, Scrum and XP, were invented for just this purpose. They are on the extreme end of the spectrum, where new requirements can be introduced daily in the case of XP and monthly in the case of Scrum. MSS also facilitates product requirements changes, mostly in response to customer and market trends, but only allows it a few times over the course of the project. RUP and TSP are roughly equivalent in this respect. Changes are allowed, but they must be justified and carefully documented.

Matrix

	RUP	MSS	TSP	XP	Scrum
a	2	2	2	3	3
b	2	2	2	2	2

a) XP and Scrum are the clear winners for this situation. This is why they were invented and are tailored to rapidly changing requirements. RUP, MSS, and TSP can accommodate this, but not as well.

b) Any process works well here.

Requirements Traceability

How important is requirements traceability and formal documentation (legal issues, contract compliance, FDA or other governmental approval, etc.)?

a) *Extremely important. Without requirements traceability and formal documentation our project is not successful.*

b) *Somewhat important. To ensure that we've delivered everything promised to the customer in our contract, we need to verify that all of the requirements have been traced from inception to completion.*

c) *Not really important. Requirements need to be met, but it is not important that they be traced. Formal documentation is not necessary, but does not hinder the product either.*

RUP places a premium on requirements traceability and formal artifacts. In fact, Rational has a product specifically tailored to requirements traceability (RequisitePro). TSP also has requirements traceability built into the process by regularly updating the required SRS (Software Requirements Specification). MSS, like TSP and RUP, has a formal SRS and Vision Document, but there is nothing prescribed in the process to trace requirements. XP and Scrum do not trace requirements. In fact, once a requirement is no longer needed, it is dropped from the story cards and the master Scrum list, with no history being kept at all.

Matrix

	RUP	MSS	TSP	XP	Scrum
a	3	2	3	1	1
b	2	2	2	1	1
c	1	2	1	3	3

c) RUP and TSP excel in this area. MSS works well. XP and Scrum can be used in this way, but it goes against the spirit of the Agile methodology.

d) RUP, MSS, and TSP work equally well. XP and Scrum still do not fit this category.

e) RUP and TSP could be tailored to not have as much formal documentation. MSS is neither good nor bad here. XP and Scrum are perfectly suited to this environment.

4 Conclusion

So, which "weapon" will you ultimately end up using to fight your battle? Well, that's a question that only you can answer, but hopefully at this point you have a much better idea of what process best suits your particular project. One of the themes of this paper has been that it is impossible to say that one particular process is "better" than another. Each process has its inherent strengths and weaknesses and is suited to particular projects, teams, and customers than others. The intent of this paper was not to glorify one method over another, but simply to provide accurate information about what industry has found to be desirable and undesirable about each. Proponents of each method would argue that their method of choice is applicable to any type of application, using any type of team, and delivering software to any type of customer. What is enlightening, however, is what industry has found through experience about each method.

A sword will hack off an arm just as well as an axe, but a cannon may not be the best weapon for hand-to-hand combat. Each weapon serves its intended purpose and they are all needed. It is the same with development processes. Make sure you understand your project and the needs of your team before deciding on a process. In short, make sure you "choose your weapon wisely."

Appendix A – Question Tally Sheet

Use this tally sheet as you answer the questions to write down the relative scores for each process. When you complete the questionnaire, add each column's scores to come up with a ranked list of processes that suit your particular project. The highest score is the most suitable, and the lowest score is the least suitable. It is important to note, however, that just because a particular process scores lower than another, it can still work for your project. The purpose of this questionnaire is not to give a definitive answer to your process dilemma, but rather to provide a starting point for your research.

Question	RUP	MSS	TSP	XP	Scrum
Team Size					
Total Developers					
Product Size and Complexity					
Competent and Experienced Developers					
Level of Hacker Sentiment					
Management Style					
Organization-Wide Processes					
New Process Adoption					
Type of Product					
Requirements Stability					
Requirements Traceability					
Totals					

Appendix B – Sample Tally Sheets

To test out my questions and relative weights, I decided to run the questionnaire against two of the more successful Studio projects in the Master of Software Engineering Program at Carnegie Mellon University. This questionnaire was filled out after-the-fact, that is, after the teams had already completed their projects using the process they had chosen at the beginning of the school year. Both teams agreed that the process they had used was very successful for them and would most likely use the same process again on a similar project.

teamMatrix – Vesmark Smartware Project

Vesmark is a financial planning company that has a patented paper-based "five-step model," in which a person can sit down and forecast his financial future. The MSE team's task was to translate the paper-based algorithms to a software system. The application was highly graphical and had to have an easy and compelling user interface. Additionally, Vesmark's business goals changed frequently and rapidly as the company sought after funding. The team decided to go with RUP in the first semester and found that the process did not work very well for them. They then switched to Scrum in the second semester to which they attribute an instant success. In the final summer semester, the team used a combination of Scrum for their management process and a modified XP as their development process. The results of their questionnaire are shown below.

Question	RUP	MSS	TSP	XP	Scrum
Team Size	1	1	3	3	3
Total Developers	2	2	2	2	2
Product Size and Complexity	1	1	1	3	3
Competent and Experienced Developers	2	2	2	2	2
Level of Hacker Sentiment	2	2	1	2	2
Management Style	2	2	3	1	1
Organization-Wide Processes	2	2	2	3	3

Question	RUP	MSS	TSP	XP	Scrum
New Process Adoption	2	2	2	2	2
Type of Product	2	2	2	2	2
Requirements Stability	2	2	2	3	3
Requirements Traceability	1	2	1	3	3
Totals	19	20	21	26	26

Interestingly, the results of the questionnaire confirm that both Sprint and XP are the best suited processes to teamMatrix's project and RUP is the least suited.

Charlatans – SEI Wrist-Camera Project

The Charlatans had a rather unique project. The clients were researchers at Carnegie Mellon's SEI (Software Engineering Institute) whose primary objective for the project was not to deliver the final product, but rather to gather research about how a team works to produce the final product. They were given a set of fixed requirements at the beginning of the project and were told that these requirements would not change and were not negotiable. The team decided to use the TSP, which is confirmed by the results of their questionnaire below.

Question	RUP	MSS	TSP	XP	Scrum
Team Size	1	1	3	3	3
Total Developers	2	2	2	2	2
Product Size and Complexity	1	1	1	3	3
Competent and Experienced Developers	1	1	3	1	1
Level of Hacker Sentiment	3	2	3	1	1
Management Style	3	3	2	2	2
Organization-Wide Processes	3	3	3	1	1
New Process Adoption	2	2	2	2	2

Question	RUP	MSS	TSP	XP	Scrum
Type of Product	2	2	2	2	2
Requirements Stability	2	2	2	2	2
Requirements Traceability	2	2	2	1	1
Totals	22	21	25	20	20

References

[Ambler 01] Ambler, Scott W., "When Does(n't) Agile Modeling Make Sense?" 2001, http://www.agilemodeling.com/essays/whenDoesAMWork.htm.

[Beck 00] Beck, Kent, *Extreme Programming Explained, Embrace Change*, Addison-Wesley, Upper Saddle River, NJ, 2000.

[Beedle] Beedle M, Devos M, Sharon Y, Schwaber K, Sutherland J, "SCRUM: An extension pattern language for hyperproductive software development," Available on the web at http://jeffsutherland.com/scrum/scrum_plop.pdf.

[Boehm 88] Boehm, Barry W., "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, 5, May 1988, pp. 61-72.

[Boehm 02] Boehm, Barry W., "Get Ready for Agile Methods, with Care," *IEEE Computer*, 1, January 2002, pp. 64-69.

[Cockburn 01] Cockburn, Alistair and Highsmith, Jim, "Agile Software Development: The People Factor," *Computer*, November 2001, pp. 131-133.

[Cockburn 02] Cockburn, Alistair, *Agile Software Development*, Addison-Wesley, Boston, MA, 2002.

[Constantine 01] Constantine, Larry, "Methodological Agility," *Software Development*, June 2001, pp. 67-69, http://www.sdmagazine.com/documents/s=730/sdm0106f/0106f.htm.

[Cusumano 95] Cusumano, Michael A. and Selby, Richard W., *Microsoft Secrets, How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, The Free Press, New York, NY, 1995.

[Himanen 01] Himanen, Pekka, *The Hacker Ethic, A Radical Approach to the Philosophy of Business*, Random House, New York, NY, 2001.

[Humphrey 95] Humphrey, Watts S., *A Discipline for Software Engineering*, Addison-Wesley, 1995.

[Humphrey 00] Humphrey, Watts S., *Introduction to the Team Software Process*, Addison-Wesley, Reading, MA, 2000.

[Jacobson 99] Jacobson, Ivar, Booch, Grady, and Rumbaugh, James, *The Unified Software Development Process*, Addison-Wesley, Upper Saddle River, NJ, 1999.

[Jeffries 01] Jeffries, Ron, Anderson, Ann, and Hendrickson, Chet, *Extreme Programming Installed*, Addison-Wesley, Upper Saddle River, NJ, 2001.

[Kruchten 00] Kruchten, Philippe, *The Rational Unified Process, An Introduction, Second Edition*, Addison-Wesley, Boston, MA, 2000.

[Kruchten 01] Kruchten, Philippe, "What Is the Rational Unified Process?" Rational Software Corporation, 2001, http://www.therationaledge.com/content/jan_01/f_rup_pk.html.

[Lindvall 02] Lindvall M, Basili V, Boehm B, Costa P, Dangle K, Shull F, Tesoriero R, Williams L, Zelkowitz M, "Empirical Findings in Agile Methods," D. Wells and L. Williams (Eds.): XP/Agile Universe 2002, LNCS 2418, pp. 197-207, 2002.

[Manifesto] Agile Alliance, "Manifesto for Agile Software Development," 2001, http://www.agilemanifesto.org/.

[Rational 01] Rational Software Corporation, "Rational Unified Process, Best Practices for Software Development Teams," *Rational Software White Paper*, TP026B, November, 2001.

[Schwaber 02] Schwaber, Ken and Beedle, Mike, *Agile Software Development with SCRUM*, Prentice-Hall, 2002.

[SEI-TSP 03] Software Engineering Institute (SEI), "The Team Software Process (TSP)," Pittsburgh, PA, July 14, 2003, http://www.sei.cmu.edu/tsp/tsp.html.

[Smith 01] Smith, John, "A Comparison of RUP® and XP," *Rational Software White Paper*, Cupertino, CA, Rational Software Corporation, 2001.

[Sutherland 03] Sutherland, Jeff, "SCRUM: Keep Team Size Under 7!" February 6, 2003, http://www.jeffsutherland.org/scrum.

[Volvo] Grahn, Goran V. and Karlsson, Boris, "Implementing RUP in an Organization—The Volvo IT Approach," Rational Sofware Whitepaper, http://www.rational.com/media/whitepapers/rup_volvo.pdf.

[Wake 02] Wake, William C., *Extreme Programming Explored*, Addison-Wesley, Upper Saddle River, NJ, 2002.

[Wheeler 01] Wheeler, David A., "More Than a Gigabuck: Estimating GNU/Linux's Size," June 30, 2001, http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html.