A SOFTWARE PROCESS ASSESSMENT MODEL AND A TOOL FOR XP@SCRUM AGILE METHOD

A Thesis Presented to the
Institute of Sciences and Engineering of Işık University
In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

by
İbrahim ULUDAĞ

IŞIK UNIVERSITY
2006

Approved for the University Committee on Graduate Studies

…….....
Prof. Dr. Hüsnü A. Erbay
Director

I certify that thesis satisfies all the requirements as a thesis for the degree of Master of Science.

…….....
Prof. Dr. Selahattin Kuru
Head of the
Computer Engineering
Department

This is to certify that I have read this thesis and that, in my opinion, it is fully adequate in scope and quality as a thesis for the degree of Master of Sciences.

……
Prof. Dr. Selahattin Kuru
Supervisor

Examining Committee Members

…….. ……

…….. ……

…….. ……

A Software Process Assessment Model and Tool for XP@SCRUM Agile Method

Abstract

In today's fast and competitive world, Agile Methods has become popular by software producers because for their high-speed, flexibility and responding to change quickly. These methods have been criticized as undisciplined way of hacking. However, these methods are disciplined processes that incorporate good engineer and management practices, albeit with extreme implementations tailored to a specific kind of environment [27]. Mark Paulk showed that organizations applying XP can reach CMM Level 2 and Level3.

These methods do not have improvement guide and capability determination. There might be differences between the organizations applying these methods.

In this thesis, I will propose a software process assessment model and a tool for proposed model. My approach is selecting an assessment model as a guide and selecting an agile method as target method.

Keywords: software process assessment, Agile Methods, CMM

Acknowledgements

Firstly, I would like to thank to my advisor, Professor Selahattin Kuru. I am grateful for his support and supervision.

Thanks to Mark Paulk from SEI for sharing his presentations about XP, CMM and Agile Methodologies.

Special thanks to Arjan C. Schokking for sharing his experiences about XP@SCRUM in Philips Research Laboratory.

Thanks to Barış Karakaya for his help.

Thanks to scrumdevelopment Yahoo Groups for sharing ideas about SCRUM.

Dedication

This thesis is dedicated to my parents who have supported me all the way since the beginning of my studies.

Also, this thesis is dedicated to my soul mate, Filiz, which has been a great source of motivation and inspiration.

Table of Contents

Abstract ..... **iii**

Acknowledgements ..... **iv**

Dedication ..... **v**

Table of Contents ..... **vi**

List of Figures ..... **ix**

List of Tables ..... **x**

1. Introduction **1**

1.1 Roadmap of Thesis 2

2. Background **3**

2.1. Software Process 3

2.2. Process Assessment 5

2.3. Agile Methodologies 6

3. Scrum **11**

3.1. Introduction to Scrum 11

3.2. Scrum Phases 12

3.3. How Does Scrum Work? 14

3.4. Scrum Roles 16

3.4.1. Scrum Master 17

3.4.2. Product Owner 17

3.4.3. Scrum Team 18

3.4.4. Stakeholders 18

3.5. Scrum Practices 18

3.5.1. Product Backlog 19

3.5.2. Daily Scrum 21

3.5.3. Sprint Planning Meeting 22

3.5.4. Sprint 22

3.5.5. Sprint Review Meeting 23

3.6. Advantages of Scrum 23

4. Extreme Programming **25**

4.1. Extreme Programming Process Model 25

4.2. XP Values: 26

4.2.1. Communication 26

4.2.2. Simplicity 27
4.2.3. Feedback 27
4.2.4. Courage 28
4.2.5. Respect 28
4.3. Core Practices 29
4.3.1. Coding Standard 29
4.3.2. Collective Code Ownership 29
4.3.3. Continuous Integration 30
4.3.4. Design Improvement 30
4.3.5. Simple Design 30
4.3.6. Small releases 31
4.3.7. Sustainable Peace 31
4.3.8. System Metaphor 31
4.3.9. Pair Programming 31
4.3.10. Planning Game 32
4.3.11. Test-Driven Development 33
4.3.12. Whole Team 33
4.4. Methodology Comparison 33
5. XP@SCRUM 35
5.1. Benefits of XP@SCRUM 37
5.2. XP@SCRUM Experiences 38
6. ISO/IEC 15504 40
6.1. Introduction to ISO/IEC 15504 40
6.2. SPICE document suite 41
6.1.1. Concepts and introductory guide 42
6.1.2. A model for process management 45
6.1.3. Rating process 48
6.1.4. Guide to conducting assessment 48
6.1.5. Construction, selection and use of assessment instruments and tools 50
6.1.6. Qualification and training of assessors 51
6.1.7. Guide for use in process improvement 53
6.1.8. Guide for use in determining supplier process capability 54
6.1.9. Vocabulary 56
7. A Model of Process Assessment for XP@SCRUM 57
7.1. Concepts and Introductory Guide 57
7.1.1. Assessment Model 58
7.2. A Model for Process Management in XP@SCRUM 58
7.2.1. Base Practices 59
7.2.2. Communication Process Category 60
7.2.3. Planning Process Category 60
7.2.4. Designing Process Category 61
7.2.5. Coding Process Category 62
7.2.6. Testing Process Category 63

7.3. Rating Process 64
7.4. Guide to Process Assessment 64
7.4.1. Reviewing the assessment scope 65
7.4.2. Selecting Process Instances 65
7.4.3. Preparing for Assessment 65
7.4.4. Information Collecting and Validation 66
7.4.5. Determining the Actual Rating 66
7.4.6. Validating Ratings 73
7.4.7. Presenting Assessment Output 73
7.5. Comparison of ISO SPICE and XP@SCRUM PCI 74

8. A Tool for XP@SCRUM PCI 75
8.1. Assessment 75
8.2. Technical Details 78

9. An Example Assessment 80
9.1. Assessment Scope 80
9.2. Preparing for Assessment 81
9.3. Information Collecting and Validation 81
9.4. Determining the Actual Rating 81
9.5. Presenting Assessment Output 84

10. Conclusion 85

References 86

List of Figures

Figure 2.1 An illustration of the software process [5] 3
Figure 2.2 Software Process Assessment [18] 5
Figure 2.3 The Planning Spectrum [8] 9
Figure 3.1 Scrum Methodology [44] 13
Figure 3.2 How Does Scrum Work? [1] 14
Figure 3.3 Input for a new Sprint [33] 15
Figure 3.4 An example of Burndown Chart [34] 21
Figure 4.1 Extreme Programming Process Model [44] 25
Figure 5.1 Overlap of XP and Scrum practices [32] 36
Figure 5.2 XP@SCRUM Model [3] 37
Figure 6.1 SPICE document suite [18] 43
Figure 6.2 Context of process assessment [18] 45
Figure 6.3 Eight assessment stages [18] 50
Figure 6.4 Software process improvement steps [18] 55
Figure 8.1 Project List 76
Figure 8.2 Add a new assessment 76
Figure 8.3 Update Assessment 77
Figure 8.4 Assessment of a Process 77
Figure 8.5 Assessment Result Chart 78
Figure 8.6 A Sample Assessment Report 78
Figure 9.1 Assessment Result of Reporting Project 84

List of Tables

Table 3.1 An example of Product Backlog [34] 20
Table 3.2 Methodology Comparison for Scrum [44] 23
Table 3.3 An example of Sprint Backlog [34] 24
Table 4.1 Methodology Comparison of Extreme Programming 34
Table 6.1 The role of the assessor in different assessment approaches [18] 52
Table 7.1 Comparison table of ISO SPICE and XP@SCRUM PCI 74

Chapter 1

Introduction

As a relatively young methodology in software engineering, Agile Methods [7] and its development practices are becoming increasingly popular, but its value is still confounded by hype and implicit, yet-to-be validated knowledge. Anecdotes of industrial teams experiencing success with partial or full implementations of these practices are abundant [22] [41]. However, organizations need a model that empirically assesses AM's efficacy.

Sim et al. challenged the software engineering community to create benchmarks – a set of tests used to compare the performance of alternative techniques [36]. Much fine work has been done on metrics for software development. The amount of literature on the subject and the process of gathering a significant set of metrics can be overwhelming to a small, informal team. Boehm and Turner [8] suggest that an informal team culture is appropriate agile methods, but this may mean that team members are less likely to be enthusiastic about formal metrics. Additionally, XP teams are often less likely to have metrics specialists on their staff due to size constraints and an avoidance of what is generally considered to be a formal, heavyweight aspect of software development. Thus, it is important to create a framework that is both informative and lightweight [43].

Even though agile methodologies are lightweight, they do have a defined process. Some agile proponents may consider this is an oxymoron because they believe that agile methods present an alternative way to a process-centered approach. They feel that their lightweight methodologies are distinctly different from the heavyweight, bureaucratic and disciplined plan-driven methodologies [8] [14]. Agile methods need a process assessment model because it must be determined whether organizations are effective in achieving their goals. XP and Scrum are the most popular agile methods. When they are combined, they provide a structure within which a customer can evolve a software product that best meets his or her needs, and can implement quality functionality incrementally to take advantage of

business opportunities. This paper proposes a software process assessment model for this agile method.

The proposed model, (XP@SCRUM PCI - Process Capability Determination and Improvement) model expects to provide set of actions to assist organizations applying XP@SCRUM in improving the way that acquire software products.

The XP@SCRUM PCI model develops a set of methodology and models covering;

- a reference model for processes and process capability
- an assessment tool for this model

The XP@SCRUM PCI model is designed to comply with the general requirements for processes in XP@SCRUM. While proposing this mode, ISO/SPICE 15504 guided as a reference Software Process Assessment Framework.

1.1 Roadmap of Thesis

My thesis is organized as follows. Part 2 gives background information for process, process assessment and agile methodologies. Part 3 explains the Scrum Agile Method. Part 4 explains the Extreme Programming. Part 5 discusses how XP and Scrum can be used together. Part 6 is a summary of ISO/IEC 15504 International Standard. Part 7 proposes an assessment model for XP@SCRUM. Part 8 explains the tool created for this model. Part 9 presents an assessment conducted by using this method and tool.

Chapter 2

Background

2.1. Software Process

As software processes have developed, the terminology has been defined successively. Sommerville defines the software process as: “The software process is the set of activities and associated results which produce a software product” [38]. This basic concept can be modeled by using an elementary process model adapted to software development [17]. This is illustrated in Figure 2.1.

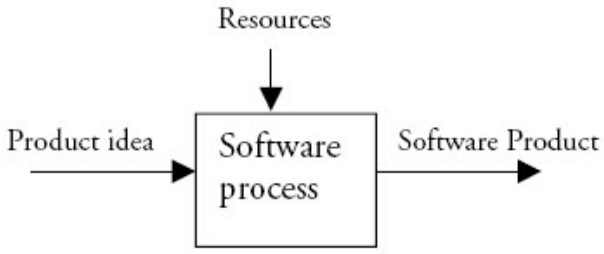

Figure 2.1 An illustration of the software process [5]

Any software process needs to at least address the following activities in some form in order to develop software [38].

- Specification. Decide what is to be developed.
- Development. Develop the product.
- Validation. Ensure the product meets the specification.
- Evolution. Handle changes in the product.

In today’s methods, this is done in many different ways. An example of software development that does not address these activities is the methods that sparked the software crisis, also known as ‘code and fix’ methods. The lack of planning and requirements in the methods, made the following problems common [38].

- Poor structure.

  After each fix the structure of the code is destroyed making subsequent fixes and additions more and more difficult.
- Inaccurate results.

  The resulting program is hardly ever what the customer desired in the first place due to the lack of requirements.
- Expensive.

  Because of poor structure and lack of planning all modifications and fixes become very expensive.

When these problems were observed, new models that addressed planning, requirements, test and modification were developed. Sommerville [38] identifies four different types of such software development models currently in professional use:

- Waterfall type

  Each activity, specification, development, validation and evolution, is executed and signed off sequentially, one by one [31].
- Evolutionary type.

  The activities are interleaved to rapidly produce prototypes of increasing complexity and correctness with regards to customer requirements [38].
- Formal transformation.

  The system is specified as a mathematical system and is then transformed into the finished product using formal mathematical transformations [20].
- Component based.

  The software system is assembled from pre-developed parts [12].

Of these, the waterfall and evolutionary types are most widely used in industry today, but the potential effectiveness of software reuse has caused much interest in component based software engineering. In the classification above, *agile methodologies* are classified as evolutionary by Sommerville. These are, however, commonly regarded as a separate type. These methodologies are very new and there is a need for more research into the area [19].

2.2. Process Assessment

According to ISO/IEC 15504, process assessment is defined as “A disciplined evaluation of an organization's software processes against the process model or variant model described in this International Standard” [18]. The figure of process assessment can be seen at the Figure 2.2.

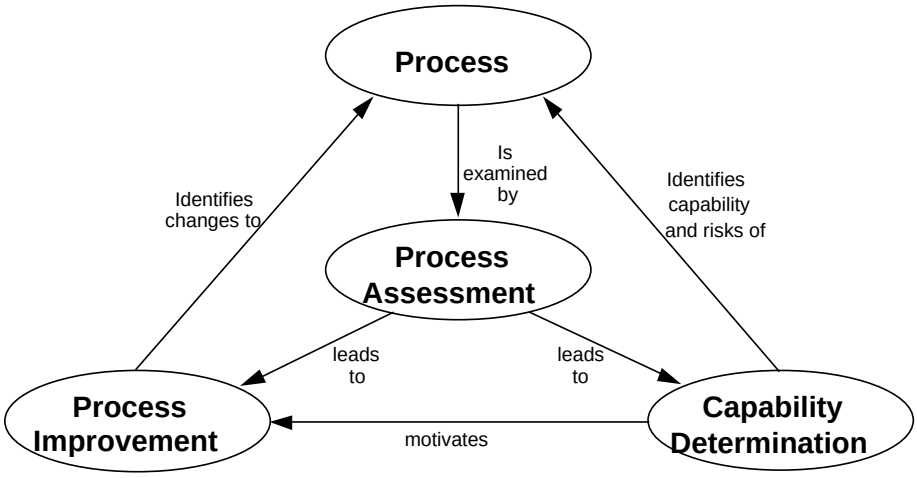

Figure 2.2 Software Process Assessment [18]

Process assessment examines the processes used by an organization to determine whether they are effective in achieving their goals. The assessment characterizes the current practice within an organizational unit in terms of the capability of the selected processes. The results may be used to drive process improvement activities or process capability determination by analyzing the results in the context of the organization's business needs, identifying strengths, weaknesses and risks inherent in the processes [18].

Within a process improvement context, process assessment provides the means of characterizing the current practice within an organizational unit in terms of the capability of the selected processes. Analysis of the results in the light of the organization's business needs identifies strengths, weakness and risks inherent in the processes. This, in turn, leads to the ability to determine whether the processes are effective in achieving their goals, and to identify significant causes of poor quality, or over runs in time or cost. These provide the drivers for prioritizing improvements to processes [18].

The framework for process assessment [18]:

- encourages self-assessment;
- takes into account the context in which the assessed processes operate;
- produces a set of process ratings (a process profile) rather than a pass/fail result;
- through the generic practices, addresses the adequacy of the management of the assessed processes;
- is appropriate across all application domains and sizes of organization.

2.3. Agile Methodologies

Agile methodologies have arisen as a reaction to the more strict processes employed during the third period of software engineering processes. The development of these occurred in parallel at the end of the 90's. The most widespread of these methodologies are listed as below [19]:

- Extreme Programming (XP)
- Scrum
- Cockburn's Crystal Family
- Open Source Software Development
- Highsmith's Adaptive Software Development
- Coad's Feature Driven Development
- DSDM (Dynamic System Development Method)
- Rational Unified Process
- Lean Programming
- Agile Modeling

These methodologies have many common aspects and in 2001 a core group of people from the agile community formulated the agile manifesto describing the most fundamental aspects of agile development. The principles of the agile manifesto are below.

The followings are the principles of the Agile Manifesto [28]:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile methodologies focus on the followings:

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

These central values that the agile community adheres to are [2] discusses as below.

First, the agile movement emphasizes the relationship and communality of software developers and the human role reflected in the contracts, as opposed to institutionalized process and development tools. In the existing agile practices, this manifest itself in close

team relationship, close working environment arrangements, and other procedures boosting team spirit.

Second, the vital objective of the software team is to continuously turn out tested working software. New releases are produced at frequent intervals, in some approaches even hourly or daily, but more usually bi-monthly or monthly. The developers are urged to keep the code simple, straightforward, and technically advanced as possible, thus lessening the documentation burden to an appropriate level.

Third, the relationship and cooperation between the developers and clients is given the preference over strict contracts, although the importance of well drafted contracts does grow at the same pace as the size of the software project. The negotiation process itself should be seen as a means of achieving and maintaining a viable relationship. From a business point of view, agile development is focused on delivering business value immediately as the project stalls, thus reducing the risks of none—fulfillment regarding the contract.

Fourth, the development group, comprising both software developers and customer representatives, should be well-informed, competent and authorized to consider possible adjustment needs emerging during the development process life-cycle. This means that the participants are prepared to make changes and that also the existing contracts are formed with tools that support and allow these enhancements to be made.

According to Highsmith and Cockburn [11], what are new about agile methods are not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability. This yields a new combination of values and principles that define an agile world view.” Boehm [8] illustrates the spectrum of different planning methods with Figure 2.3, in which hackers are placed at one end and the so called inch-pebble ironbound contractual approach at the opposite end.

Cockburn defines the core of agile software development methods as the use of light-but-sufficient rules of project behavior and the use of human- and communication-oriented

rules. The agile process is both light and sufficient. Lightness is a means of remaining maneuverable. Sufficiency is a matter of staving in the came [11]. He proposes the following “sweet spots” the presence of which in software development work enhances the prospects for a successful project outcome:

1. Two to eight people in one room
2. Communication and community
3. Onsite usage experts
4. Short and continuous feedback cycles
5. Short increments
6. One to three months, allows quick testing and repairing
7. Fully automated regression tests
8. Unit and functional tests stabilize code and allow continuous improvement
9. Experienced developers
10. Experience speeds up the development time from 2 to 10 times compared to slower team members

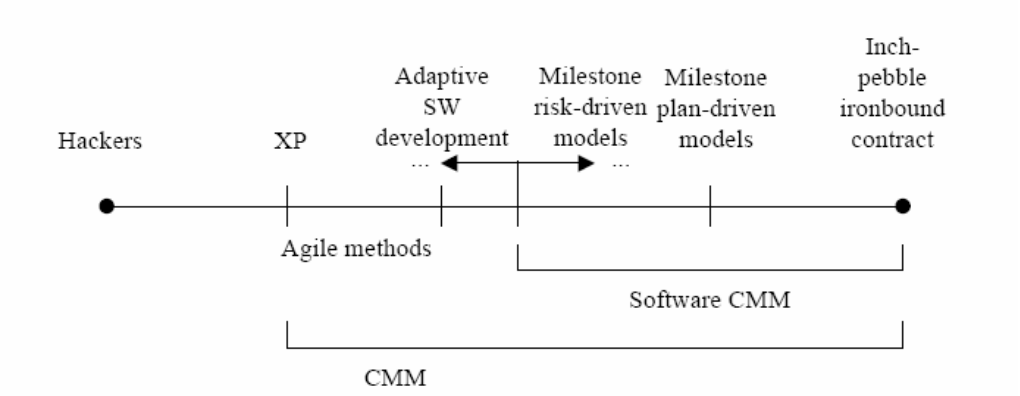

Figure 2.3 The Planning Spectrum [8]

Miller [25] gives the following characteristics to agile software processes from the fast delivery point of view, which allow shortening the life-cycle of projects:

1. Modularity on development process level
2. Iterative with short cycles enabling fast verifications and corrections
3. Time-bound with iteration cycles from one to six weeks

4. Parsimony in development process removes all unnecessary activities
5. Adaptive with possible emergent new risks
6. Incremental process approach that allows functioning application building in small steps
7. Convergent (and incremental) approach minimizes the risks
8. People-oriented. I.e. agile processes favor people over processes and technology
9. Collaborative and communicative working style

The basic principles of agile methods comprise an unforgiving honesty of working code, effectiveness of people working together with goodwill, and focus on teamwork. A set of common sense approaches emerging from agile software development processes have been suggested by Ambler [4] as follows:

1. people matter
2. less documentation is possible
3. communication is a critical issue
4. modeling tools are not as useful as usually thought
5. big up-front design is not required

When software development is incremental (small software releases, with rapid cycles), cooperative (customer and developers working constantly together with close communication), straightforward (the method itself is easy to learn and to modify, well documented), and adaptive (able to make last moment changes), the development is called agile [2].

Chapter 3

Scrum

This chapter gives information about Scrum and how it is implemented. This information is based on the book “Agile Software Development with Scrum” [33].

3.1. Introduction to Scrum

In today’s fast-paced, fiercely competitive world of commercial new product development, speed and flexibility are essential. Companies are increasingly realizing that the old, sequential approach to developing new products simply won’t get the job done. Instead, companies in Japan and the United States are using a holistic method: as in rugby, the ball gets passed within the team as it moves as a unit up the field.

Scrum implements an empirical approach based in process control theory. The empirical approach reintroduces flexibility, adaptability, and productivity into system development.

Scrum is an iterative, incremental process for developing any product or managing any work. It produces a potentially shippable set of functionality at the end of iteration. Its attributes are listed as below [1]:

1. Scrum is an agile process to manage and control development work.
2. Scrum is a wrapper for existing engineering practices.
3. Scrum is a team-based approach to iteratively, incrementally develop systems and products when requirements are rapidly changing
4. Scrum is a process that controls the chaos of conflicting interests and needs.
5. Scrum is a way to improve communications and maximize co-operation.
6. Scrum is a way to detect and cause the removal of anything that gets in the way of developing and delivering products.
7. Scrum is a way to maximize productivity.

8. Scrum is scalable from single projects to entire organizations. Scrum has controlled and organized development and implementation for multiple interrelated products and projects with over a thousand developers and implementers.
9. Scrum is a way for everyone to feel good about their job, their contributions, and that they have done the very best they possibly could.

Scrum naturally focuses an entire organization on building successful products. Without major changes -often within thirty days - teams are building useful, demonstrable product functionality. Scrum can be implemented at the beginning of a project or in the middle of a project or product development effort that is in trouble.

Scrum is a set of interrelated practices and rules that optimize the development environment, reduce organizational overhead, and closely synchronize market requirements with iterative prototypes. Based in modern process control theory, Scrum causes the best possible software to be constructed given the available resources, acceptable quality, and required release dates. Useful product functionality is delivered every thirty days as requirements, architecture, and design emerge, even when using unstable technologies.

3.2. Scrum Phases

Scrum has the following three phases [44]:

- **Pregame**
  - Planning: Create the Product Backlog (A prioritized list of requirements)
  - Architecture : Use backlog to create high level design of architecture
- **Game**
  - Development Sprints: The software is created in these sprints.
    - Develop: Defining changes needed for the implementation of backlog requirements into packets, opening the packets, performing domain analysis, designing, developing, implementing, testing, and documenting the changes. Development consists of the micro process of discovery, invention, and implementation.

        - Wrap: Closing the packets, creating a executable version of changes and how they implement backlog requirements.
        - Review: All teams meeting to present work and review progress, raising and resolving issues and problems, adding new backlog items. Risk is reviewed and appropriate responses defined.
        - Adjust: Consolidating the information gathered from the review meeting into affected packets, including different look and feel and new properties.
- **Postgame**
    - Closure: Make preparation for release, including final documentation and testing.

These phases can be summarized in the Figure 3.1.

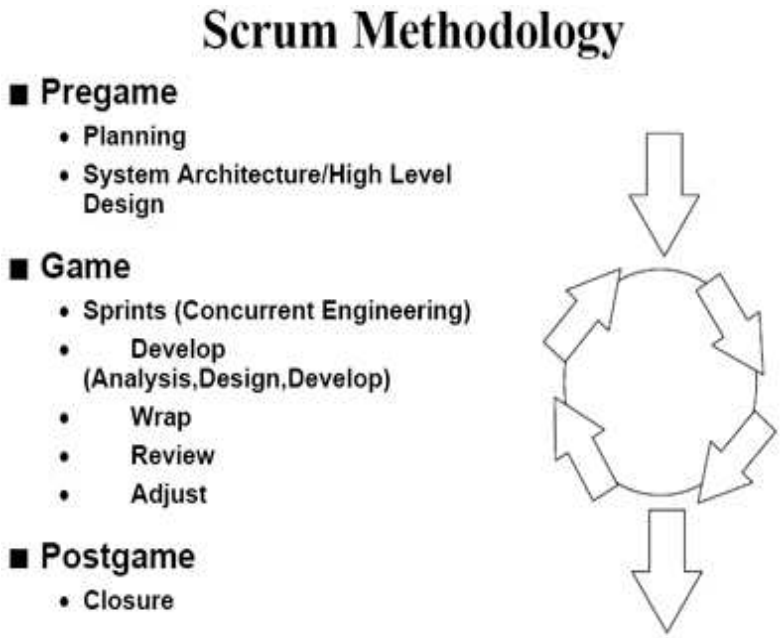

Figure 3.1 Scrum Methodology [44]

3.3. How Does Scrum Work?

Scrum is based on the concept of iterations. There are two nested iterations at the end of which a feasible product appears. The inner iteration is done every day and is called a scrum. The external iteration is done every month (30 days- which, according to the proponents of Scrum is the ideal - neither long nor short- for viable software to be developed). This is illustrated in the Figure 3.2.

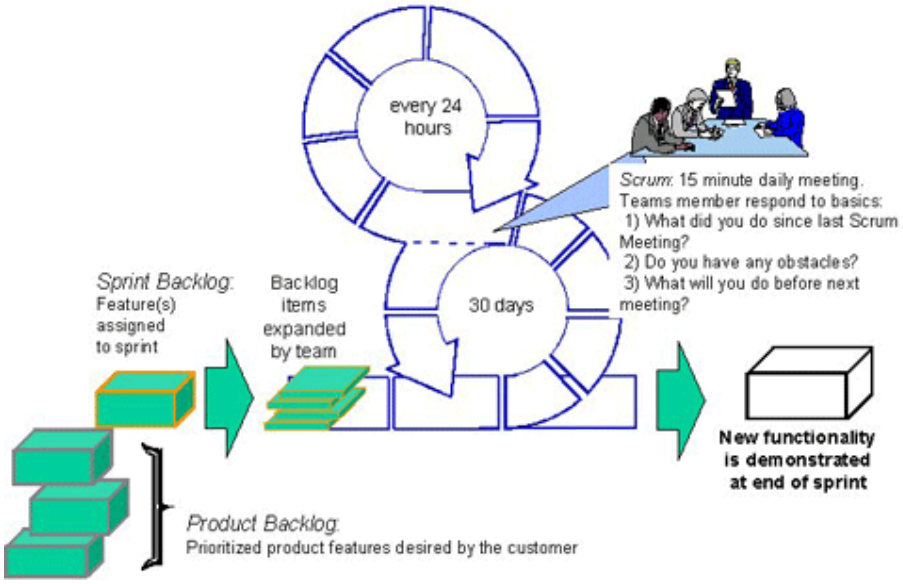

Figure 3.2 How Does Scrum Work? [1]

The Product Backlog is the listing of the things that the system should include and address, including functionality, features and technology. This is a prioritized list of all product requirements. Product backlog is never finalized. Rather, it emerges and evolves along with the product. Items that have high priority on the Product Backlog are the ones that are most desired. Product Backlog content can come from anywhere: users, customers, sales, marketing, customer service, and engineering can all submit items to backlog. However, only the Product Owner can prioritize the backlog. The Product Owner effectively decides the order in which things are built.

Small, cross-functional teams perform all development (Scrum Teams). These teams take on as much Product Backlog as they think they can turn into an increment of product functionality within a thirty-day iteration, or **Sprint.** Every Sprint must finish by delivering new executable product functionality. Architecture and design emerge across multiple Sprints, rather than being completely during the first Sprints. Figure 3.3 explains how a new Sprint is formed.

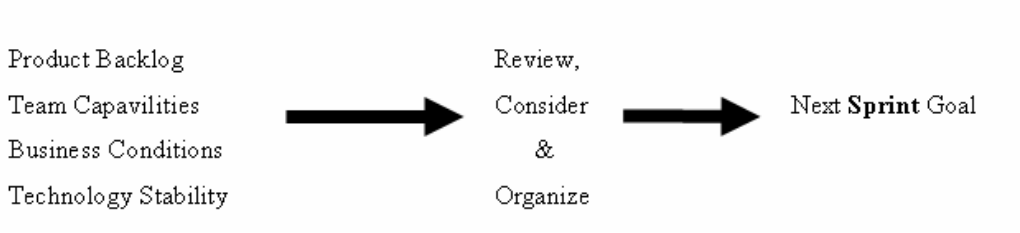

Figure 3.3 Input for a new Sprint [33]

Multiple teams can develop product increments in parallel, all teams working the same Product Backlog. The Scrum Teams are self-organizing and fully autonomous. They are constrained only by organization's standards and conventions, and by the Product Backlog that they have selected. How the Product Backlog will be turned into a product increment is up to team decisions. The team maintains a list of tasks to perform during each Sprint that is called **Sprint Backlog**.

Scrum relies on team initiative and integrity. During the sprint, a management representative (**Scrum Master**) enforces Scrum practices and helps the team to make decisions or acquire resources as needed. The team must not be disturbed or given direction by anyone by outside of it while it is in a Sprint.

The Scrum Team meets daily for a short status meeting, called **Daily Scrum.** At the Daily Scrum, Progress is reviewed and impediments identified for removal by management. The Daily Scrum is an excellent place to observe how much progress a team is making.

At the end of the Sprint, the team gets together with management at a **Sprint Review Meeting** to inspect the product increment the team has built. They either build on what was

developed, scavenge it, or throw it away. However, the pressure to build on what's been developed is high. The thirty day Sprint duration ensures that the worst that happens is that thirty days are lost should the team prove unable to develop any useful product functionality.

After the product increment is inspected, management often rearranges the Product Backlog to take account of what the team has accomplished. The Product Backlog has more meaning when viewed in light of the partially developed product. Sometimes so many backlogs are built that management selects an earlier release schedule. In this case, the next Sprint can be used to release the product.

Once the Product Backlog has been stabilized, the team again selects top priority Product Backlog for the next Sprint. The team then goes through another iteration of work, pushing through another Sprint. This cycle continues until the product, based on **Empirically Managing** cost, time, functionality, and quality – is deemed potentially releasable. Release Sprints are then devised to bring the product to release-readiness.

Scrum is straightforward. By stripping away inappropriate and cumbersome management practices, Scrum leaves only the essence of work. Scrum leaves a team free to go it, to work its heart out and build the best product possible. Although the Scrum process seems simple and skeletal, it provides all necessary management and control to focus developers and quickly build quality products.

In Scrum, there is no formal project planning phase. There aren't any Pert charts. There are no roles and individual assignments. The team is able to get on with its work and build valuable product increments anyway. The team self-organizes from a dispirited group of individuals waiting for instructions into a team that takes the initiative and acts.

3.4. Scrum Roles

Scrum has four roles as listed below.

1. Scrum Master
2. Product Owner

3. Scrum Team
4. Stake Holders

These roles are better defined then the ones in XP which is due to the fact that Scrum focuses on management and control where XP focuses on engineering practices so well defined roles are more important. That is also why most of the time a person will only have a single role and none are left out in a typical project [32].

3.4.1. Scrum Master

Scrum Master is the person that manages the Scrum process in an organization. The Scrum master is responsible for the success of Scrum.

The Scrum Master is a new management role introduced by Scrum. The Scrum Master is responsible for ensuring that Scrum values, practices, and rules are enacted and enforced.

The Scrum Master represents management and the team to each other. At the Daily Scrum, the Scrum master listens closely to what each team member reports. He or she compares what progress has been made to what progress was expected, based on Sprint goals and predictions made during previous Daily Scrum.

The Scrum Master works with the customer and management to identify and institute a Product Owner. The Scrum master works with management to form Scrum teams. The Scrum Master then works the Product Owner and the Scrum teams to create Product Backlog for a Sprint. The Scrum Master works with the Scrum teams to plan and initiate the Sprint. During the Sprint, the Scrum Master conducts all Daily Scrums, and is responsible for ensuring that impediments are promptly removed and decisions are promptly made. The Scrum Master is also responsible for working with management to gauge process and reduce backlog.

3.4.2. Product Owner

Product owner is the customer representative. Product owner can be only a person not a committee. He/she is responsible for managing and controlling the Product Backlog.

3.4.3. Scrum Team

A team commits to achieving a Sprint Goal. The team is accorded full authority to do whatever it decides is necessary to achieve the goal.

The Scrum Master meets with the Scrum team and reviews the Product Backlog. The Scrum Team commits to turn a selected set of Product Backlog into a working product. The Scrum team makes this commitment every Sprint. The team has full authority to do whatever is necessary to do so. It is only constrained by organizational standards and conventions.

Every individual has their own strengths and weaknesses, comes from a unique background, and is trained and gains skills through a unique education and job history. Pair Programming enables to gain the strengths of team dynamics.

A team selects the amount of Product Backlog and establishes the Sprint Goal. No third party can commit a person or team to do work.

It is important to equip a team with the best possible tools. Open environments allow people to communicate more easily.

3.4.4. Stakeholders

Users, management, sponsors are the stakeholders of the project [29]. The stakeholders can join the Scrum meetings to see the status of the project but they cannot speak in the meeting. They can give their opinions about the product features and perform functional tests.

3.5. Scrum Practices

Scrum practices provide practical methods to complete the Scrum development life cycle.

3.5.1. Product Backlog

The Product Backlog represents everything that anyone interested in the product or process has thought is needed or would be a good idea in product. It is a list of all features, functions, technologies, enhancements, and bug fixes that constitute the changes that will be made to the product for future releases. Table 3.1 shows an example of a Product Backlog.

Product backlog is initially incomplete, just an initial list of all things that the product or system needs. The first Product Backlog may be a list of requirements that is gleaned from a vision document garnered from a brainstorming sessions, or derived from a marketing requirements document. Sources of Product Backlog are formal or informal as the hosting organization. To get the first Sprint going, Product Backlog only needs to contain enough requirements to drive a thirty-day Sprint. A Sprint can start from only concepts and wish list.

The product backlog emerges from this initial list as the product and the customer's understanding of their needs emerge and evolve. Backlog is dynamic. Management repeatedly changes it to identify what the product requires to be appropriate competitive, and useful. As long as a product exists, Product Backlog also exists.

Product Backlog is sorted in order of priority. Top priority Product Backlog drives immediate development activities. Higher priority backlog is clearer and has more detailed specification than lower priority backlog. Better estimates are made based on the greater clarity and increased detail.

A Burndown chart shows the amount of work remaining across time. The Burndown chart is an excellent way of visualizing the correlation between the amount of work remaining at any point in time and the progress of the project Team(s) in reducing this work. The intersection of a trend line for work remaining and the horizontal axis indicates the most probable completion of work at that point in time. A Burndown Chart reflecting this is shown in Figure 3.4. This allows to "what if" the project by adding and removing functionality from the release to get a more acceptable date or extend the date to include

Table 3.1 An example of Product Backlog [34]

Backlog Description	Init. Est.	Adj. Factor	Adj. Est.							
Title Import				1	2	3	4	5	6	7
Project Selection or new	3	0,2	3,6	3,6	0	0	0	0	0	0
Template Backlog for new projects	2	0,2	2,4	2,4	0	0	0	0	0	0
Create P. B. workseet with formatting	3	0,2	3,6	3,6	0	0	0	0	0	0
Create S. B. worksheet with formatting	3	0,2	3,6	3,6	0	0	0	0	0	0
Display tree view of product backlog, releases, sprints	2	0,2	2,4	2,4	0	0	0	0	0	0
Sprint-1	13	0,2	15,6	16	0	0	0	0	0	0
Create a new window containing product backlog template	3	0,2	3,6	3,6	3,6	0	0	0	0	0
Create a new window containing sprint backlog template	2	0,2	2,4	2,4	2,4	0	0	0	0	0
Burndown window of product backlog	5	0,2	6	6	6	0	0	0	0	0
Burndown window of sprint backlog	1	0,2	1,2	1,2	1,2	0	0	0	0	0
Display tree view of product backlog, releases, sprints	2	0,2	2,4	2,4	2,4	0	0	0	0	0
Display burndown for selected or release	3	0,2	3,6	3,6	3,6	0	0	0	0	0
Sprint-2	16	0,2	19,2	19	19	1,2	0	0	0	0
Automatic recalculating of values and tools	3	0,2	3,6	3,6	3,6	3,6	0	0	0	0
As changes are made to backlog in secondary window update burndown graph on main page	2	0,2	2,4	2,4	2,4	2,4	0	0	0	0
Hide/automatic display of burndown window	3	0,2	3,6	3,6	3,6	3,6	0	0	0	0
Insert sprint capability	2	0,2	2,4	2,4	2,4	2,4	0	0	0	0
Insert Release capability	1	0,2	1,2	1,2	1,2	1,2	0	0	0	0
Owner assigned capability and columns opt.	2	0,2	2,4	2,4	2,4	2,4	0	0	0	0
Print burndown graphs	1	0,2	1,2	1,2	1,2	1,2	0	0	0	0
Sprint-3	14	0,2	16,8	17	17	17	0	0	0	0
Duplicate incomplete backlog without affecting totals	5	0,2	6	6	6	6	6	6	6	6
Note capability	6	0,2	7,2	7,2	7,2	7,2	7,2	7,2	7,2	7,2
What-if release capability on burndown graph	15	0,2	18	18	18	18	18	18	18	18
Trend capability on burndown server	2	0,2	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4
Publish capability for entire project, publishing it as HTML web pages	11	0,2	13,2	0	0	13	13	13	13	13
Future Sprints	39	0,2	46,8	34	34	34	34	34	34	34
Release-1				85	70	65	47	47	47	47

more functionality. The Burndown Chart is the collision of reality (work done and how fast it's being done) with what is planned, or hoped for [34].

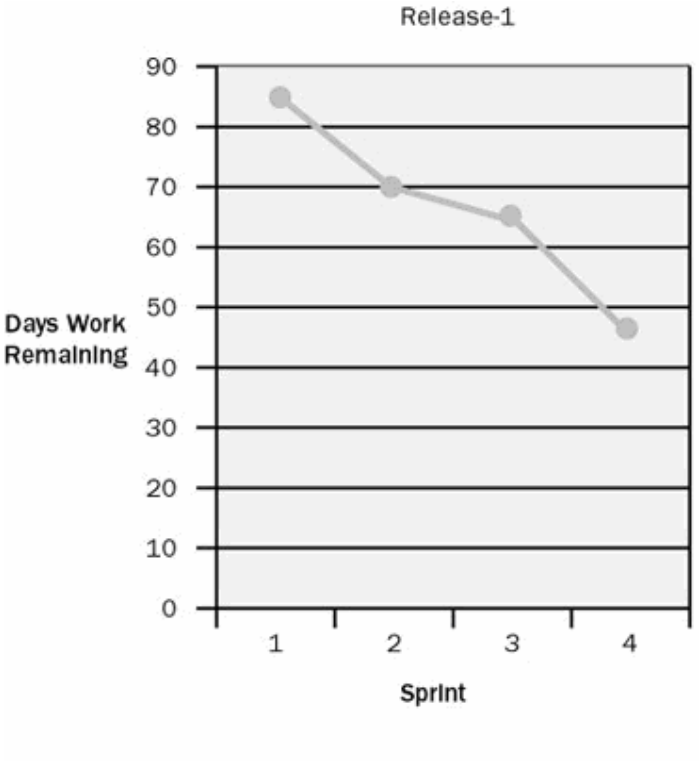

Figure 3.4 An example of Burndown Chart [34]

3.5.2. Daily Scrum

Each Scrum Team meets daily for a 15-minute status meeting called the daily Scrum. During the meeting, the team explains what it has accomplished since the last meeting, what is going to do before the next meeting, and what obstacles are in its way. Attending a Daily Scrum is easier and more informative than reading a report and Daily Scrums have additional benefit of being a boon for the team as well as for its managers.

During the Daily Scrum, only one person talks at a time. Everyone reports his or her status. Scrum Master asks everyone three questions.

1. What have you done since the last Scrum?
2. What will you do between now and the next Scrum?
3. What got in your way of doing work?

The Daily Scrum is not a design session and should not turn into a working session.

3.5.3. Sprint Planning Meeting

Customers, users, management, the Product Owner and the Scrum Team determine the next Sprint goal and functionality at the Sprint Planning meeting. The team then devises the individual tasks that must be performed to build the product increment. (Figure 3.3)

To start the meeting, the Product Owner presents the top priority Product Backlog. Having selected the Product Backlog, a Sprint Goal is crafted. The reason for having a Sprint Goal is to give the team some wiggle room regarding the functionality.

After establishing the Sprint goal, the team determines what work will have to be performed. The team compiles a list of tasks to complete to meet the Sprint Goal. These tasks are detailed pieces of work needed to convert the Product Backlog into working software. Tasks should be enough detailed so that each task takes roughly four to sixteen hours to finish. This task list is called Sprint Backlog. The team self-organizes to assign and undertake the work in the Sprint Backlog (Table 3.3). The team may have to define an initial architecture or create designs before it can fully delineate the rest of the design. In such a case, the team should define the initial investigation, design, and architecture work in as much detail as possible, and reminders for work that will probably have to be done once the investigation or design has been completed. Only the team can change its Sprint Backlog. The Sprint Backlog is a highly visible, real time picture of the work that the team plans to accomplish the Sprint, and it belongs solely to the team.

3.5.4. Sprint

A team is let loose for the thirty day Sprint. The team has committed to the goal and accepted the responsibility of building a product increment that meets the goal. It has the authority to act as it as it sees fit. No person outside the team can change the scope or nature of the work the team is doing during a Sprint. No one is allowed to add more functionality or technology to the Sprint. No one can tell the team how to proceed in its work.

Every product development project is constrained by four variables, (1) time available (2) cost, in people and resources, (3) delivered quality, and (4) delivered functionality. A Sprint greatly fixes the first three variables.

3.5.5. Sprint Review Meeting

The Sprint Review meeting is a four-hour informational meeting. During the meeting, the team presents to management, customers, users, and the Product Owner the product increment that it has built during the Sprint.

3.6. Advantages of Scrum

The Scrum methodology, on the other hand, is designed to be quite flexible throughout. It provides control mechanisms for planning a product release and then managing variables as the project progresses. This enables organizations to change the project and deliverables at any point in time, delivering the most appropriate release [44].

The comparison of Scrum with other methodologies is shown in the Table 3.2.

Table 3.2 Methodology Comparison for Scrum [44]

	Waterfall	**Spiral**	**Iterative**	**Scrum**
Defined Process	Required	Required	Required	Planning & Closure
Final Product	Determined During Planning	Determined During Planning	Set During Project	Set During Project
Project Cost	Determined During Planning	Partially Variable	Set During Project	Set During Project
Completion Date	Determined During Planning	Partially Variable	Set During Project	Set During Project
Responsiveness to environment	Planning Only	Planning Primarily	At end of each iteration	Throughout
Team flexibility, Creativity	Limited - cookbook appr.	Limited - cookbook appr.	Limited - cookbook appr.	Unlimited During Iterations
Knowledge Transfer	Training prior To project	Training prior to project	Training prior To project	Teamwork during project
Probability of Success	Low	Medium Low	Medium	High

Table 3.3 An example of Sprint Backlog [34]

Task Description	Originator	Responsible	Status	1	2	3	4	5	6	7	8	9	10	11	12
Meet to discuss the goals and features for Sprint 3-6	Danielle	Danielle/Sue	Completed	20	0	0	0	0	0	0	0	0	0	0	0
Move Calculations out of Crystal reports	Jim	Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Get KEG Data		Tom	Completed	12	0	0	0	0	0	0	0	0	0	0	0
Analyze KEG Data - Title		George	In Progress	24	24	24	24	12	12	12	12	12	12	12	12
Analyze KEG Data - Parcel		Tim	Completed	12	12	12	12	12	4	4	4	0	0	0	0
Analyze KEG Data - Encum brance		Josh	In Progress							12	10	10	10	10	10
Analyze KEG Data – Contact		Danielle	In Progress	24	24	24	24	12	10	8	6	6	6	6	6
Analyze KEG Data – Facilities		Allen	In Progress	24	24	24	24	12	10	10	10	10	10	10	10
Define & Build Database		Barry/Dave	In Progress	80	80	80	80	80	80	60	60	60	60	60	60
Validate the size of KEG Database		Tim	Not Started												
Look at KEG Data on the G:\		Dave	In Progress	3	3	3	3	3	3	3	3	3	3	3	3
Confirm Agreement with KEG		Sue	Not Started												
Confirm KEG Staff Availability		Tom	Not Started	1	1	1	1	1	1	1	1	1	1	1	1
Switch JDK 1.3.1. Run all tests		Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Store PDF files in a structure		Jacquie	Completed	8	0	0	0	0	0	0	0	0	0	0	0
TopLink. Cannot get rid of netscape parser		Richard	Completed	4	0	0	0	0	0	0	0	0	0	0	0
Build test data repository		Barry	In Progress	10	10	10	10	10	10	10	10	8	8	8	8
Move application and database to Qual		Richard	Completed	4	4	4	4	4	4	4	0	0	0	0	0
Set up Crystal Environment		Josh	Completed	2	2	2	2	1	1	1	0	0	0	0	0
Test app in Qual		Sue	In Progress												20
Definining Sprint Goal required for solution in 2002		Lynne	In Progress	40	40	40	40	40	40	40	38	38	38	38	38
Reference tables for import process		Josh	In Progress												
Build standard import exception process		Josh	In Progress									12	12	12	10
Handle multiple file imports on same page		Jacquie	Disregarded												
Migrate to CruiseControl Servlet		Allen	Not Started	4	4	4	4	4	4	4	4	4	4	4	4

Chapter 4

Extreme Programming

Extreme Programming (XP) was conceived and developed to address the specific needs of software development conducted by small teams in the face of vague and changing requirements. This new lightweight methodology challenges many conventional tenets, including the long-held assumption that the cost of changing a piece of software necessarily rises dramatically over the course of time. XP recognizes that projects have to work to achieve this reduction in cost and exploit the savings once they have been earned [6].

4.1. Extreme Programming Process Model

The Extreme Programming process model is shown in the Figure 4.1.

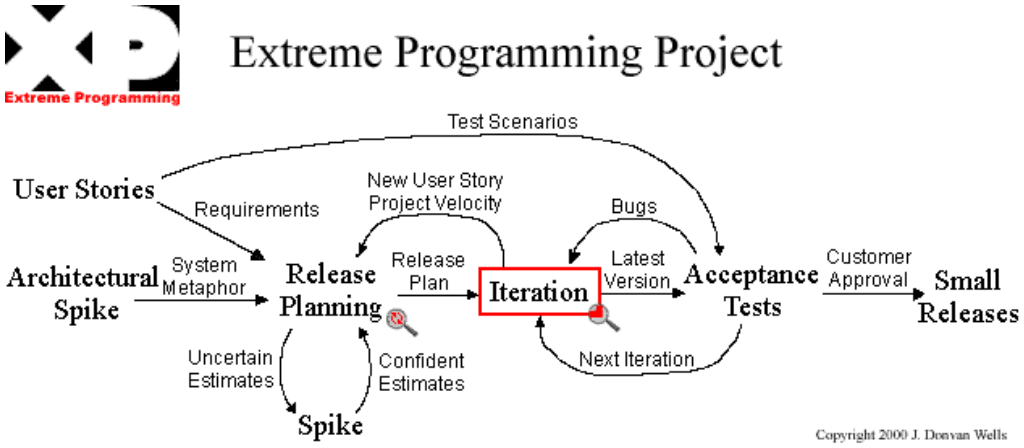

Figure 4.1 Extreme Programming Process Model [44]

In Extreme Programming, the customer identifies the needs by writing user stories. The customer prioritizes the stories in the planning game meeting. The software is created with fast iterations. After each iteration; a working and tested software is created. The tests

include unit tests performed by developers and functional tests performed by customers. The software is created by using simple design, and refactoring.

4.2. XP Values:

Extreme Programming initially recognized just four values but a new value was added in the second edition of Extreme programming explained. The five values are [13]:

- Communication
- Simplicity
- Feedback
- Courage
- Respect (the latest value)

4.2.1. Communication

Extreme Programming techniques can be viewed as methods for rapidly building and disseminating institutional knowledge among members of a development team. The goal is to give all developers a shared view of the system which matches the view held by the users of the system. To this end, Extreme Programming favors simple designs, metaphor, collaboration of users and programmers, frequent verbal communication and feedback [13].

Communication is accomplished by the followings [23]:

- Collaborative workspaces
- Co-location of development and business space
- Paired development
- Frequently changing pair partners
- Frequently changing assignments
- Public status displays
- Short standup meetings
- Unit tests, demos and oral communication, not documentation

4.2.2. Simplicity

Extreme Programming encourages starting with the simplest solution and refactoring to better ones. The difference between this approach and more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, next week, or next month. Proponents of XP acknowledge the disadvantage that this can sometimes entail more effort tomorrow to change the system; their claim is that this is more than compensated for by the advantage of not investing in possible future requirements that may change before they become relevant. Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed. Related to the previous value, "communication", simplicity in design and coding should improve the (quality of) communication. A simple design with very simple code can be easily understood by every programmer in the team [13].

Simplicity encourages the followings [23]:

- Delivering the simplest functionality that meets business needs
- Designing the simplest software that supports the needed functionality
- Building for today and not for tomorrow
- Writing code that is easy to read, understand, maintain and modify

4.2.3. Feedback

Within Extreme Programming, feedback is related to different dimensions of the system development [13]. These dimensions are listed as below:

- Feedback from the system: by writing unit tests the programmers have direct feedback from the state of the system after implementing changes.
- Feedback from the customer: The functional tests are written by the customer and the testers. They will get concrete feedback about the current state of their system. This review is planned once in every two or three weeks so the customer can easily steer the development.
- Feedback from the team: When customers come up with new requirements in the planning game the team directly gives an estimation of the time that it will take to implement.

Feedback is closely related to communication and simplicity. Flaws in the system are easily communicated by writing a unit test that proves a certain piece of code will break. The direct feedback from the system tells programmers to recode this part. A customer is able to test the system periodically according to the functional requirements.

Feedback is provided by the followings actions [23]:

- Aggressive iterative and incremental releases
- Frequent releases to end users
- Co-location with end users
- Automated unit tests
- Automated functional tests
- Do the right thing in the face of opposition
- Do the practices required to succeed

4.2.4. Courage

The Extreme Programming doctrine of "Courage in system development" can be best explained by a couple of practices [13]. One is the commandment to always design and code for today and not for tomorrow. This is an effort to avoid getting bogged down in design and requiring a lot of effort to implement anything else. Courage enables developers to feel comfortable with refactoring their code when necessary. This means reviewing the existing system and modifying it so that future changes can be implemented more easily. Another example of courage knows when to throw code away. Every programmer has experienced getting stuck on a complex problem in their own design and code after working on it all day, then coming back the next day with a clear and fresh view and rapidly solving the problem in half an hour.

4.2.5. Respect

In Extreme Programming the Respect value has different perspectives. There's respect for other team members because, even with short cycles and continuous integration, programmers never commit changes that break compilation, that make existing unit tests fail, or that otherwise delay the work of their peers. There's respect for one self in always

striving for high quality and seeking for the best design for the solution at hand through refactoring [13].

4.3. Core Practices

Extreme Programming has 12 core practices. These are listed as below.

1) Coding standard
2) Collective code ownership
3) Continuous integration
4) Design improvement
5) Simple design
6) Small releases
7) Sustainable pace
8) System metaphor
9) Pair programming
10) Planning game
11) Test driven development
12) Whole team

4.3.1. Coding Standard

Coding standard is an agreed upon set of rules that the entire development team agree to adhere to throughout the project [14]. By following a coding standard, all of the code looks like if it was written by a one person. The team can choose standard written by other person/company or creates own coding standard.

The benefits of Coding Standard are:

- Code is understandable by all team
- Supports Collective Code Ownership

4.3.2. Collective Code Ownership

Collective code ownership means that everybody is responsible for all the code; this, in turn, means that everybody is allowed to change any part of the code [14].

The benefits of Collective Code Ownership are:

- Decrease the time for development and fixing bug
- Provides continuity of the project
  If a team member leaves the team, any team member can continue his/her task
- Reduce duplicate code

4.3.3. Continuous Integration

Continuous Integration is a software development practice where members of a team integrate their work frequently; usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. This reduces the integration problems [15].

4.3.4. Design Improvement

Extreme Programming focuses on delivering business value in each iteration. To accomplish this over the course of the whole project, the software must be well-designed. The alternative would be to slow down and ultimately get stuck [21].

The refactoring process focuses on removal of duplication (a sure sign of poor design), and on increasing the "cohesion" of the code, while lowering the "coupling". High cohesion and low coupling have been recognized as the hallmarks of well-designed code for at least thirty years. The result is that XP teams start with a good, simple design, and always have a good, simple design for the software. This lets them sustain their development speed, and in fact generally increase speed as the project goes forward [42].

4.3.5. Simple Design

XP team develops the software by using the simplest thing that works. This means implementing only the required features. Simple design enables to handle changes that will occur in the future.

4.3.6. Small releases

Small releases enable to create the software which the customer wants. At the end of each release, the customer gives a feedback to the XP team.

The benefits of small releases can be listed as below:

- The customers gets a working product at the end of each release
- The customer can change the scope, add/remove functions

4.3.7. Sustainable Peace

Extreme Programming teams are in it for the long term. They work hard, and at a pace that can be sustained indefinitely. This means that they work overtime when it is effective, and that they normally work in such a way as to maximize productivity week in and week out [21].

4.3.8. System Metaphor

System Metaphor is a vision of what the program is and how the program works. At its best, the metaphor is a simple evocative description of how the program works, such as "this program works like a hive of bees, going out for pollen and bringing it back to the hive" as a description for an agent-based information retrieval system [42]. By system metaphor, a programmer can make his/her decision about the program.

4.3.9. Pair Programming

Pair programming means code is produced by two developers sitting next by next and looking to the same monitor.

Many customers think that pair programming is unnecessary and increases the budget. However there are many researches showing pair programming increases the quality of the product, decreases the development time [10].

The benefits of pair programming [10] can be listed as below:

- many mistakes get caught as they are being typed in rather than in QA test or in the field (continuous code reviews);

- the end defect content is statistically lower (continuous code reviews);
- the designs are better and code length shorter (ongoing brainstorming and pair relaying);
- the team solves problems faster (pair relaying);
- the people learn significantly more, about the system and about software development (line of-sight learning);
- the project ends up with multiple people understanding each piece of the system;
- the people learn to work together and talk more often together, giving better information flow and team dynamics;
- people enjoy their work more

4.3.10. Planning Game

The main planning process within Extreme Programming is called the planning game.

The planning process is divided into two parts [14]:

Release Planning: This is focused on determining what requirements are included in which release and when it's going to be delivered. The customers and developers are both part of this. Release Planning consists of three phases:

- **Exploration Phase:** In this phase the customer will give all his requirements for the system. These will be written down on user story cards.
- **Commitment Phase:** Within the commitment phase business and development will commit themselves to the functionality that will be included and the date of the next release.
- **Steering Phase:** In the steering phase the plan can be adjusted, new requirements can be added and or existing requirements can be changed or removed.

Iteration Planning: This plans the activities and tasks of the developers. In this process the customer is not involved. Iteration Planning also consists of three phases:

- **Exploration Phase:** Within this phase the requirement will be translated to different tasks. The tasks are recorded on task cards.
- **Commitment Phase:** The tasks will be assigned to the programmers and the time it takes to complete will be estimated.

- **Steering Phase:** The tasks are performed and the end result is matched with the original user story.

4.3.11. Test-Driven Development

Extreme Programming is obsessed with feedback, and in software development, good feedback requires good testing. Top XP teams practice "test-driven development", working in very short cycles of adding a test, then making it work [42].

There are two types of tests:

- **Unit Test:** Unit tests are automated tests written before code is written. These tests are written to check if the functionality of the written code will pass. Before the release, all unit tests must pass.
- **Functional Tests:** Functional tests are the acceptance test made by customer.

4.3.12. Whole Team

All the contributors to an XP project sit together, members of one team. This team must include a business representative -- the "Customer" -- who provides the requirements, sets the priorities, and steers the project. It is best if the Customer or one of her aides is a real end user who knows the domain and what is needed. The team will of course have programmers. The team may include testers, who help the Customer define the customer acceptance tests. Analysts may serve as helpers to the Customer, helping to define the requirements. There is commonly a coach, who helps the team keep on track, and facilitates the process. There may be a manager, providing resources, handling external communication, coordinating activities. None of these roles is necessarily the exclusive property of just one individual: Everyone on an XP team contributes in any way that they can. The best teams have no specialists, only general contributors with special skills [42].

4.4. Methodology Comparison

The methodology comparison is shown in the Table 4.1.

Table 4.1 Methodology Comparison of Extreme Programming

	Waterfall	**Spiral**	**Iterative**	**XP**
Defined Process	Required	Required	Required	Not Exists
Final Product	Determined During Planning	Determined During Planning	Set During Project	Set During Project
Project Cost	Determined During Planning	Partially Variable	Set During Project	Set During Project
Completion Date	Determined During Planning	Partially Variable	Set During Project	Set During Project
Responsiveness to environment	Planning Only	Planning Primarily	At end of each iteration	Throughout
Team flexibility, Creativity	Limited - cookbook approach	Limited - cookbook approach	Limited - cookbook approach	Unlimited During Iterations
Knowledge Transfer	Training prior To project	Training prior to project	Training prior to project	Paired Programming
Probability of Success	Low	Medium Low	Medium	High

Chapter 5

XP@SCRUM

Scrum and Extreme Programming provide complementary practices and rules. They overlap at the planning game (XP) and Sprit planning (Scrum). Both encourage similar values, minimizing otherwise troublesome disconnects between management and developers. Combined, they provide a structure within which a customer can evolve a software product that best meets his or her needs, and can implement quality functionality incrementally to take advantage of business opportunities. Following are several shared practices that facilitate this functionality [21]:

Iterations: All work is done iteratively, with the customer being able to steer and direct the project in every iteration.

Increments: the team produces an increment of the customer's highest-priority functionality in every iteration. If desired, the customer can direct the developers to turn these increments into live, operational functionality at any time.

Emergence: Only that functionality that the customer has selected for the next iteration is considered and built. The customer doesn't pay for functionality that he or she might not select, and the developers don't have to code, debug, and maintain irrelevant code.

Self-organization: The customer says what he or she wants; development determines how much they can develop during an iteration and figures out the tasks to do so.

Collaboration: Business and engineering collaborate about how best to build the product and what the product should do between iterations.

Scrum doesn't have any engineering practices, wrapping and using those at the organization where it is implemented. When these engineering practices are weak, overall productivity is lessened [21].

XP doesn't have any management practices. XP tells management where it needs them, but offers few insights into maximizing value [21].

Both methods complement each other very well and can be combined to address the issues with the lack of methods for dealing with low level issues in Scrum and high level planning issues in XP. Also the only real overlap between the two methodologies is the planning game which was directly lifted from Scrum by XP to give it at least rudimentary process estimation and tracking ability. This overlap causes no problems since it is implementation is the same in both methodologies. All the other practices the two methodologies have are exclusive to them. They are shown in Figure 5.1. Here the red circle describes XP with its practices listed to the right while the blue circle depicts Scrum with its practices to the left. The purple section is the overlap and as described above only holds the planning game [32].

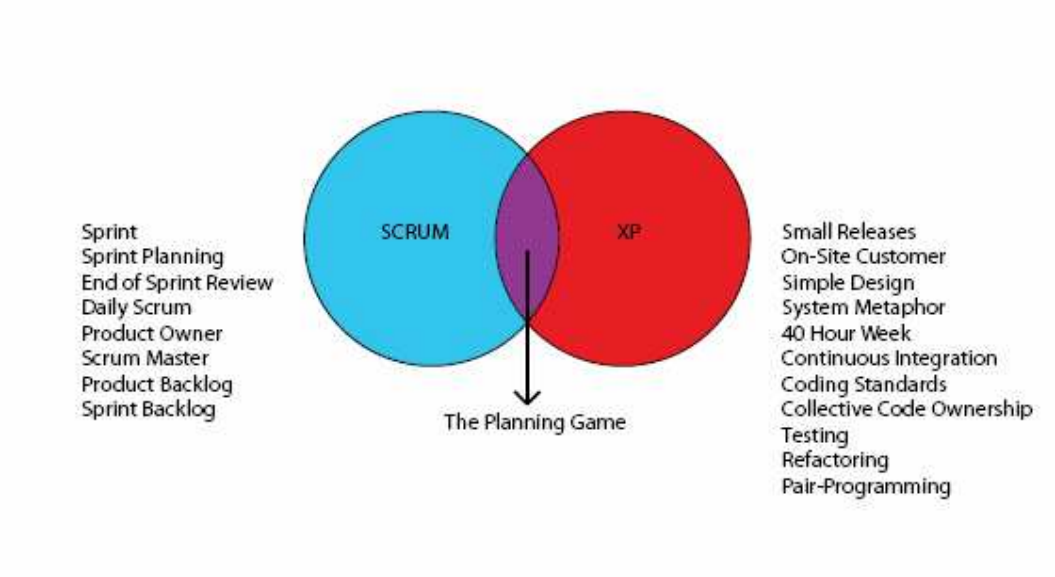

Figure 5.1 Overlap of XP and Scrum practices [32]

And the Figure 5.2 shows how XP and Scrum is implemented together.

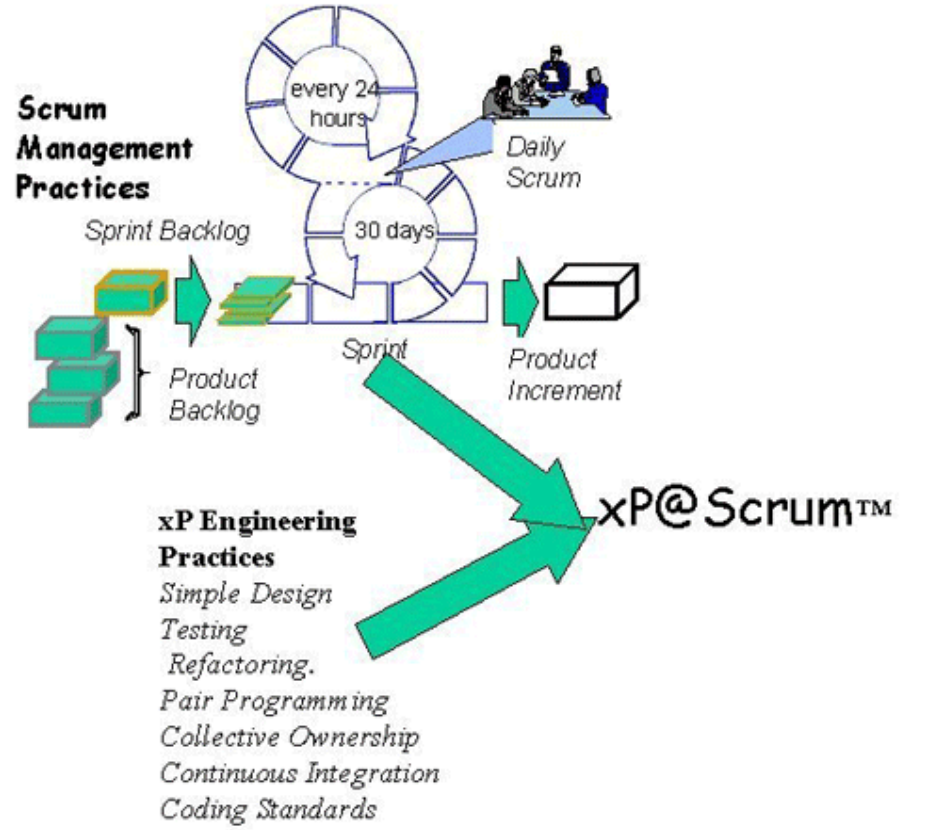

Figure 5.2 XP@SCRUM Model [3]

5.1. Benefits of XP@SCRUM

Benefits of XP@SSCRUM can be listed as below [3]:

- The agile management and control mechanisms of Scrum are applicable for any type of project, including business initiatives that consist of multiple, simultaneous software development, business development, re-engineering, marketing, support, and implementation projects. XP@SCRUM projects fit within the overall management framework of these initiatives.
- XP@SCRUM projects realize the full benefits of self-organization; teams are iteration (or Sprint) goal directed, rather than story directed.
- When Extreme Programming projects are wrapped by Scrum, they become scalable and can be run simultaneously by non-collocated teams.

- Scrum implements in a day; Extreme Programming can be gradually implemented within the Scrum framework.

5.2. XP@SCRUM Experiences

Philips Research has selected the combination of Extreme Programming and Scrum as the base of their software development process. Firstly they applied XP and then they used XP with Scrum. While applying only XP, there had problems as below [40]:

1. XP does not give you much help regarding documentation, modeling, and the use of UML and design patterns.
2. Newly hired software engineers had to be instructed where the architecture is addressed in XP. The term is only briefly mentioned in the 12 practices. Our current belief is that a good introductory course regarding XP is required for each new employee. They must be instructed that writing tests first has everything to do with architecture and design: it forces to look at the code from the viewpoint of its user which is at a higher abstraction level. And that it offers a practical way to obtain the principle of low coupling and high cohesion. This is because tests must be fully self-contained. Furthermore refactoring, as explained by Martin Fowler [16], allows to not only addressing architecture at the beginning of the project (as in the waterfall model) but offers the opportunity to evolve the software architecture in a cost-effective sense. Certainly, the principle design-as-you-go compared to code-and-fix must be explained from a practical context
3. XP did not help us regarding introduction of XP in an organization, how to optimize or enhance the way of working, and how to interact with your management. Also as with most managers, the word extreme scared them off. Maybe we may propose the word excellent here. We are aware of course, that XP would not be what it is now, if XP would not have such a provocative name
4. We could not convince all our customers to provide us with unambiguous requirements in the form of acceptance tests. The best we could get from them were scenario's to execute, with some related sentences regarding the expected outcome. Defining automated tests for all requirements at the user level in the multimedia domain though, is a challenge on its own. We now schedule meetings at the end of

the iteration in which the team demonstrates the currently implemented functionality to the customer and discusses together whether it is accepted or not.

5. An iteration length of two weeks worked out to be too short for our small teams (1-5 in size) in order to be able to add significant functionality. We are now using iteration lengths of one calendar month, which also introduces a nice rhythm in our projects: at the beginning of the month new requirements have to be identified and at the end of the month these requirements have to be accepted. This is easily remembered even without inspecting our calendars.
6. We felt the need to not only enter functional requirements in the list of user stories but also nonfunctional ones, like moving from one tool to another, a major refactoring or a particular required document. After all, it is the customer that pays for them and we want them to become visible and approved. We now also add problem reports and change requests to this list.

After applying XP with Scrum, they have solved the problems numbered 3 through 6.

Chapter 6

ISO/IEC 15504

This chapter summarizes the international standard for process assessment ISO/IEC 15504 [18] which is a reference model for this thesis.

6.1. Introduction to ISO/IEC 15504

ISO/IEC 15504 (SPICE) is a major international initiative to support the development of an International Standard for Software Process Assessment. The project has three principal goals [37]:

- To develop a working draft for a standard for software process assessment
- To conduct industry trials of the emerging standard
- To promote the technology transfer of software process assessment into the software industry world-wide

The standard is designed to provide assessment results that are repeatable, objective, comparable within similar contexts, and able to be used for either process improvement or process capability determination is used by an organization to determine whether they are effective in achieving their goals. The assessment characterizes the current practice within an organizational unit in terms of the capability of the selected processes. The results may be used to drive process improvement activities or process capability determination by analyzing the results in the context of the organization's business needs, identifying strengths, weaknesses and risks inherent in the processes.

The first goal was achieved on June 1995 when the version 1 (draft standard) was released. By the normal process of development of international standards, the SPICE documents have been published as ISO/IEC TR 15504:1998 - Software Process Assessment. WG10 is continuing the work to the ultimate goal, full international standard.

There are a lot of people involved in SPICE development from over 20 countries. The international efforts are coordinated by five international technical centers. This arrangement has brought together software, standards, process and many other developers and academics around the world. The ultimate goal of this international community is to develop a consistent and validated framework for software process assessment.

Why an international standard is needed? To improve quality and productivity, management needs to somehow measure the process used in development. Process assessment can be a strong and effective driver for process improvement. An international standard will provide the following benefits to software industry:

- Software suppliers will submit to just one process assessment scheme (presently numerous schemes are used)
- Software development organizations will have a tool to initiate and sustain a continuous process improvement
- Program managers will have a means to ensure that their software development is aligned with, and supports, the business needs of the organization

Practically it means that companies can get much better situation in the competitive world-wide markets when they use internationally standardized software process assessment framework. Companies participating to the SPICE development ensure that they are at the forefront of this technology when it will reach the international standard [30].

SPICE can be used in various contexts. Before using SPICE, the organization must first define the key determinant of why SPICE is needed. There are three choices:

- To understand the software process used
- To support process improvement
- To support process capability determination

6.2. SPICE document suite

SPICE provides a set of documents, which are used as a framework for the assessment of software process. Organizations can use these documents in various phases of production,

for example in planning, managing, monitoring, controlling and improving acquisition, supply, development, operation, evolution and support of software.

Basically software process assessment examines the selected processes whether they are effective in achieving their goals, which is done by determining the capability of the selected processes. This structured approach for software process assessment helps an organization to improve its processes or to determine its capability for certain requirement, or to determine supplier's capability for certain requirement.

Process assessment provides information of the capability of the selected processes. Analysis results, from business point of view, identify strengths, weakness and risks inherent in the processes [37]. By this, analyzers are able to determine whether the processes are effective, and to identify significant causes of poor quality, or over runs in time or cost. After recognizing these kinds of issues, managers can prioritize improvements to processes.

Process capability determination analyses the proposed capability of selected processes against a target process capability profile. By this, it tries to find out the risks involved in a project, if the project is run with the analyzed processes.

The document suite of SPICE contains nine different documents, which can be used in process assessment. The relationship of these documents can be seen at the Figure 6.1.

6.1.1. Concepts and introductory guide

This part of this International Standard provides overall information on the concepts of software process assessment and its use in the two contexts of process improvement and process capability determination. It describes how the parts of the suite fit together, and provides guidance for their selection and use. It explains the requirements contained within this International Standard, and their applicability to the conduct of an assessment, to the construction and selection of supporting tools, and to the construction of extended processes.

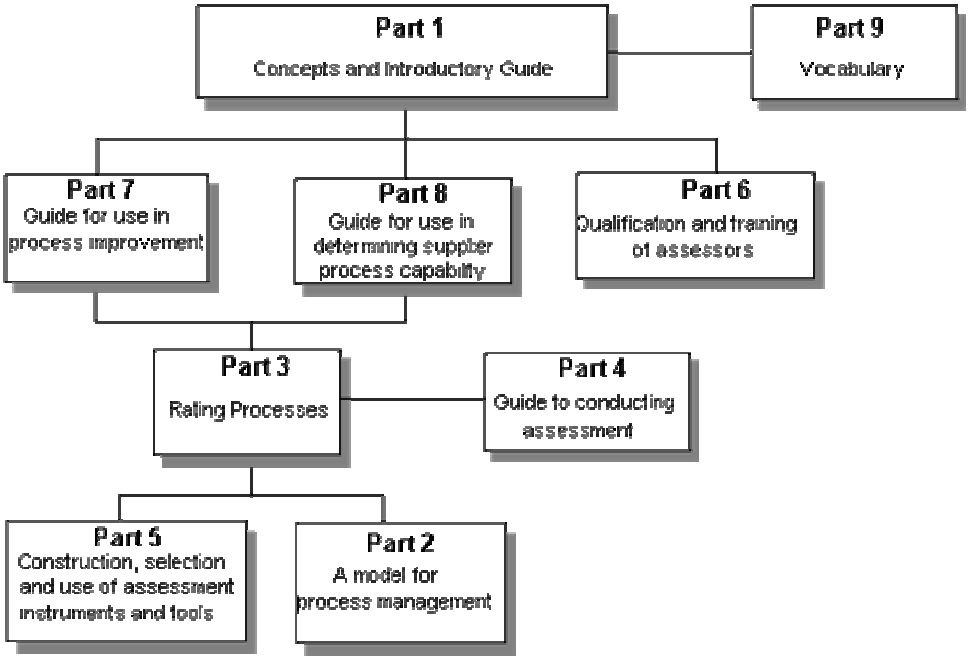

Figure 6.1 SPICE document suite [18]

Figure 6.2 shows how process assessment and process capability determination affects to process improvement. Basically a process is examined with an assessment, which leads to process capability determination and process improvement. Capability determination identifies the capability and risks of a process, and process improvement identifies the changes, which should be made to the process. Software capability determination generally motivates an organization to do process improvement [30].

The benefits arising from the use of this suite of documents include:

For **acquirers**:

- An ability to determine the current and potential capability of a supplier's software processes

For **suppliers**:

- An ability to determine the current and potential capability of their own software processes
- A ability to define areas and priorities for software process improvement
- A framework that defines a road map for software process improvement

For **assessors**:

- A framework that defines all aspects of conducting assessments

Processes, categorized into five process categories in the model, are described below:

- **Customer-Supplier** - processes that directly impact the customer, support development and transition of the software to the customer, and provide for its correct operation and use
- **Engineering** - processes that directly specify, implement, or maintain a system and software product and its user documentation
- **Project** - processes which establish the project, and co-ordinate and manage its resources to produce a product or provide a service which satisfies the customer
- **Support** - processes which enable and support the performance of the other processes on a project
- **Organization** - processes which establish the business goals of the organization and develop process, product, and resource assets which will help the organization achieve its business goals

Context of a Process Assessment

An assessment is carried out by assessing selected processes against the process model defined in part 2 of this International Standard. This two-dimensional model consists of a set of process-specific base practices and a set of generic practices. The generic practices apply across all processes. The generic practices are grouped into common features and capability levels that may be used to determine how well the process is managed. The assessment output includes a set of process capability level ratings for each process instance assessed. The context of a process assessment is summarized in Figure 6.2.

An assessment is supported by an assessment instrument, or set of instruments, constructed according to part 5 of this International Standard. The process assessment is carried out either by a team with at least one qualified assessor who has the competence described in part 6; or, on a continuous basis using suitable tools for data collection and verified by a qualified assessor.

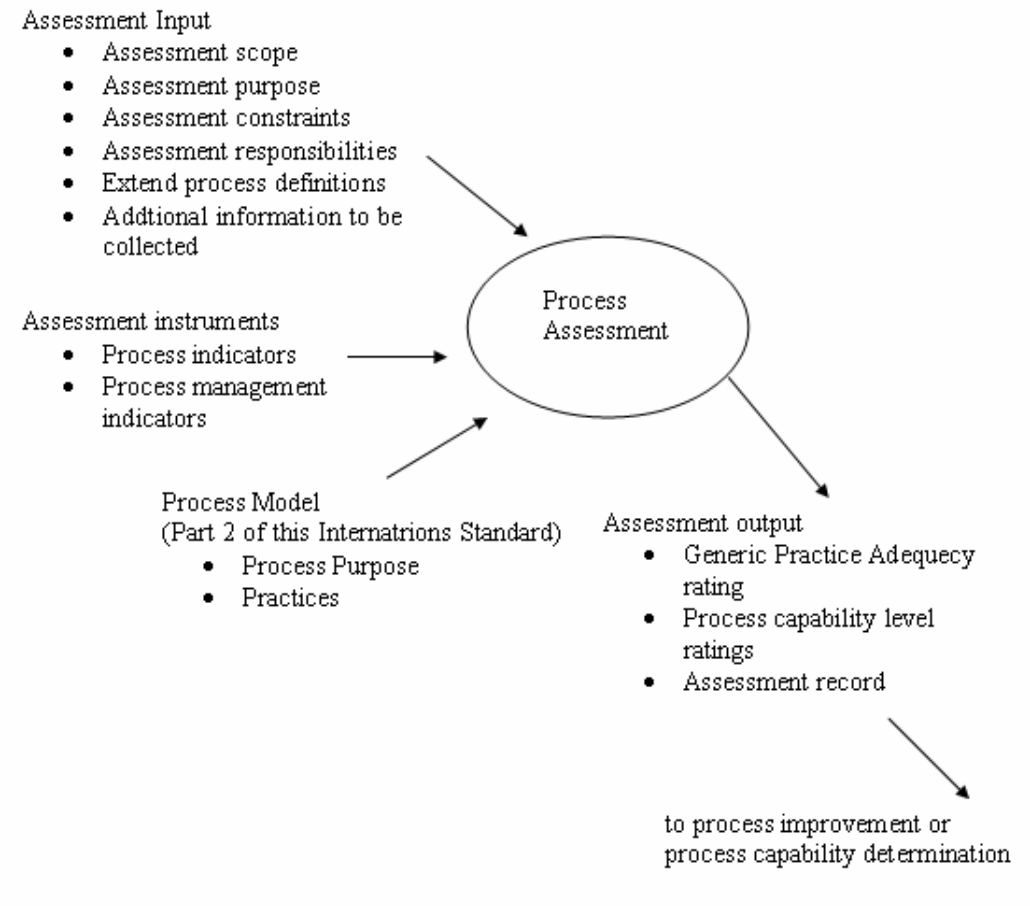

Figure 6.2 Context of process assessment [18]

Tools to support process assessment

An assessment instrument is a tool, or set of tools, used during the performance of an assessment to assist the assessor in obtaining reliable, consistent and repeatable results.

6.1.2. A model for process management

Document 2 provides the set of practices fundamental to good software engineering. This document defines various processes which can be used in various phases of production, named acquire, supply, development, support, evolve, and operate.

The model includes a set of practices, named basic practices and generic practices. Basic practices, grouped into processes and process categories, are essential activities of a

specific process, while generic practices, applicable to any process, represent the activities necessary to manage a process and improve its capability to perform.

Process capability levels, common features, and generic practices are used in evolving process capability. A capability level basically consists of a set of common features (sets of activities), which provide enhancement in the capability of performing a process. Compared to the predecessors, each level provides a major enhancement in capability of the performance of a process.

Capability levels provide two benefits:

- Acknowledging dependencies among the practices of a process
- Help an organization to identify which improvements it might perform first

Capability levels are named as followed:

- Level 0 – Not performed

  The Not-Performed level has no common features. There is general failure to perform the base practices in the process. There are no easily identifiable work products or outputs of the process.

- Level 1 – Performed informally

  Base practices of the process are generally performed. The performance of these base practices may not be rigorously planned and tracked. Performance depends on individual knowledge and effort. Work products of the process testify to the performance. Individuals within the organization recognize that an action should be performed, and there is general agreement that this action is performed as and when required. There are identifiable work products for the process.

- Level 2 – Planned and tracked

  Performance of the base practices in the process is planned and tracked. Performance according to specified procedures is verified. Work products conform to specified standards and requirements.

The primary distinction from the Performed-Informally Level is that the performance of the process is planned and managed and progressing towards a well-defined process

- Level 3 – Well defined
  Base practices are performed according to a well-defined process using approved, tailored versions of standard, documented processes.

  The primary distinction from the Planned-and-Tracked Level is that the process of the Well-Defined Level is planned and managed using an organization-wide standard process.

- Level 4 – Quantitatively controlled
  Detailed measures of performance are collected and analyzed. This leads to a quantitative understanding of process capability and an improved ability to predict performance. Performance is objectively managed. The quality of work products is quantitatively known.

  The primary distinction from the Well-Defined Level is that the defined process is quantitatively understood and controlled.

- Level 5 – Continuously improving
  Quantitative process effectiveness and efficiency goals (targets) for performance are established, based on the business goals of the organization. Continuous process improvement against these goals is enabled by quantitative feedback from performing the defined processes and from piloting innovative ideas and technologies.

The primary distinction from the Quantitatively-Controlled Level is that the defined process and the standard process undergo continuous refinement and improvement, based on a quantitative understanding of the impact of changes to these processes.

6.1.3. Rating process

This document suite of SPICE is used in defining the minimum set of requirements for conducting a software process assessment. These requirements are used to ensure that the outputs of the assessment are consistent, repeatable and representative of the process instances assessed.

A process assessment is practically done by assessing selected processes against the process model defined in document 2. The output of the assessment provides a set of capability level ratings for each process instance assessed.

This document is primarily addressed to the qualified assessor and other people, such as the sponsor of the assessment, who need to assure themselves that the requirements have been met. It may also be very valuable for developers of assessment methods and tools.

As part of the SPICE, this document establishes the requirements for a software process assessment, for rating, analyzing and profiling an assessment, and defines the circumstances under which assessment results are comparable.

This document provides an assessment framework which:

- Encourages self-assessment
- Takes into account the context in which the assessed processes operate
- Produces a set of process ratings (a process profile) rather than a pass/fail result
- Through the generic practices, addresses the adequacy of the management of the assessed processes
- Is appropriate across all application domains and sizes of organization

6.1.4. Guide to conducting assessment

Process assessment basically means just collecting information describing the current capability of an organization’s processes. It is initiated if there is a need to determine and/or improve the capability of these processes. SPICE document 4 provides guidance on interpreting the requirements set out in part 3 primarily for the use in a team-based assessment.

Although this guidance is directed at conducting a team-based assessment, the principles for rating processes can be used in a continuous, tool-based assessment. Nevertheless, in a continuous assessment the data collection is somehow different.

This document is primarily aimed at the followings:

- The assessment team for preparing the assessment
- The participants in the assessment for understanding the assessment and interpreting the results
- All staff within organizations for understanding the details and benefits of performing process assessment
- Tool and method developers for developing tools or methods supporting the process assessment model

Process assessment

Figure 6.2 describes how a process assessment can be initiated by the need for process improvement or process capability determination. Assessment input is collected with the help of assessment instrument, and the process model is used in assessment. Finally the output is used for process improvement or process capability determination.

Assessment guide

The assessment contains eight stages, described in Figure 6.3 below. The stages are:

- Assessment input review
- Process instance selection
- Preparations
- Information collection and verification
- Determination of actual ratings
- Determination of derived ratings
- Rating validation
- Presenting the assessment output

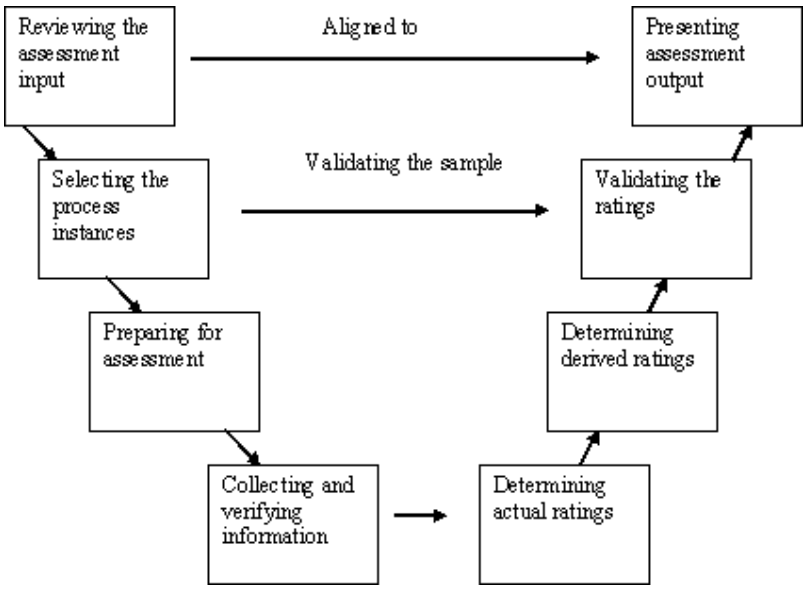

Figure 6.3 Eight assessment stages [18]

6.1.5. Construction, selection and use of assessment instruments and tools

SPICE document 5 establishes the requirements for constructing an assessment instrument. In addition, it provides guidance on selection and usability characteristics associated with various types of assessment instruments.

An assessment instrument is defined as a tool (or set of tools) which is basically used in evaluation of the adequacy or existence of practices. An assessment instrument is needed to provide a consistent set of indicators as discriminators to help judge how well the practices have been implemented. An assessment instrument provides also a mechanism for recording the collected information.

This document:

- Sets out the minimum requirements to be met in the construction of an assessment instrument
- Defines a set of indicators to be included in an assessment instrument
- Provides guidance on the selection, construction and usability of assessment instruments

This part of the International Standard is directed to the following people:

- Those responsible for the design and construction of assessment instruments, e.g. methodology providers, tool suppliers, assessors
- Assessors and assessment teams with responsibility for the selection and procurement of appropriate assessment instruments
- Assessors, sponsors or other parties responsible for assessing conformance of an assessment instrument to these requirements

Construction of an assessment instrument

It is not required in this standard that an assessment instrument should take any particular form or format. It can be, for example, a paper-based instrument containing forms, questionnaires or checklists, or it can be, for example, a computer-based instrument such as a spreadsheet, a data base system or an integrated CASE tool.

"Regardless of the form of the assessment instrument, its main objective is to help an assessor to perform an assessment in a consistent and repeatable manner, reducing assessor subjectivity and ensuring the validity, usability and comparability of the assessment results".

"All indicators incorporated into an assessment instrument shall be traceable to a corresponding process, generic practice, or base practice in the process model in part 2 of this international standard or to a practice in an extended process".

6.1.6. Qualification and training of assessors

This part of the International Standard defines the initial and ongoing qualification of assessors and provides guidance for the preparation and qualification of assessors to perform software process assessments. It describes mechanisms that may be used to demonstrate assessor competence and to validate an assessor's education, training and experience.

The guidance in this document is applicable to an organizational unit or a sponsor of an assessment wishing to select or specify the type of assessors to perform either self-assessments or independent assessments.

The guidance is also applicable to the identification and demonstration of the competencies necessary for the performance of assessments, and to the process of obtaining those competencies.

The role of the assessor, as described in part 4 of this International Standard, is to assess the capability of the software process of an organizational unit in a constructive and objective manner. The assessment should be focused on the process and not the people implementing the process. The role varies depending on the assessment approach as shown in Table 6.1.

Table 6.1 The role of the assessor in different assessment approaches [18]

Self-assessment approach	Independent assessment approach
Is task and people oriented.	Is task oriented.
Guides the assessment.	Controls the assessment.
Delivers an approach.	Delivers a rating.
Promotes discussion.	Regulates discussion.
Works with projects.	Works separately from projects.
Uses organizational unit's business goals.	May be indifferent to organizational unit's business goals.
Influences through results obtained, relationships established and expertise.	Influences through position and expertise.
Seeks compliance and commitment.	Determines process adequacies.
Is like being a change agent.	Is like being an auditor.

The result of the assessment obviously depends on the skilled judgment of the assessors. “The achievement of an acceptable level of consistency, repeatability and reliability of results relies on competent assessors with appropriate skills, experience, and knowledge of the software process, of the model for processes described in document 2, and of the conduct of assessment and rating described in parts 3 and 4”.

A qualified assessor usually acts as a team leader for the assessment team. This person is in responsibility of ensuring that other team members have the right blend of specialized knowledge and assessment skills. This qualified assessor has to provide the necessary guidance and lead to the team, and help to moderate the judgments and ratings made by other team members to ensure the consistency of the results.

This document practically describes the assessor competencies and appropriate education, training and experience. The document also introduces mechanisms used in demonstrating the competence and validating the education, training and experience.

In addition to technical skills, assessors should have also certain personal skills, like diplomacy, persistence and judgment.

6.1.7. Guide for use in process improvement

Document 7 acts as a guide on using process assessment to understand the current state of processes, and to create and prioritize the improvement plans. The document is primarily aimed at the management of an organization considering a software process improvement programme, members of improvement teams, software engineers, and external consultants.

This process improvement guide includes the following:

- An overview of process improvement – the factors which drive software process improvement and general principles which underpin it
- A methodology for process improvement – an eight step model for improving software processes within a continuous improvement cycle
- Cultural issues – aspects of organizational culture that are critical for successful process improvement
- Management – software process improvement from a management perspective including an overall framework for process measurement

This guidance, used in software assessment for process improvement, covers the following:

- Invoking a software process assessment

- Using the results of a software process assessment
- Measuring software process effectiveness and improvement effectiveness
- Identifying improvement actions aligned to business goals
- Using the process model in document 2 as a route map for improvement
- Cultural issues in the context of software process improvement
- Dealing with management issues for software process improvement

The guidance provided by this document, does not presume specific organizational structures, management philosophies, software life cycle models or software development methods. The guidance and principles are appropriate for different business needs, application domains and sizes of organization, so that they may be used by all types of software organizations to guide their improvement activities.

Figure 6.4 illustrates the steps for continuous software process improvement using the components of SPICE. A comprehensive process improvement programme may identify improvement goals to be attained over several iterations of the improvement cycle.

6.1.8. Guide for use in determining supplier process capability

This part of the International Standard provides guidance on how to utilize process assessment for the purposes of process capability determination [18].

A process capability determination (PCD) is a systematic assessment and analysis of selected software processes within an organization, carried out with the aim of identifying the strengths, weaknesses and risks associated with deploying the processes to meet a particular specified requirement.

Process capability determination is applicable in a variety of situations; the specified requirement may involve a new or an existing task, a contract or an internal undertaking, a product or a service, or any other requirement which is to be met by deploying an organization's software processes.

This guidance is intended to be applicable across all software application domains, over all software organizational structures, within any software customer-supplier relationship, and to any organization wishing to determine the process capability of its own software processes.

This guide is primarily aimed at:

- The sponsor who initiates the process capability determination
- The organization whose process capability is to be determined
- The assessment team
- Tool and method developers

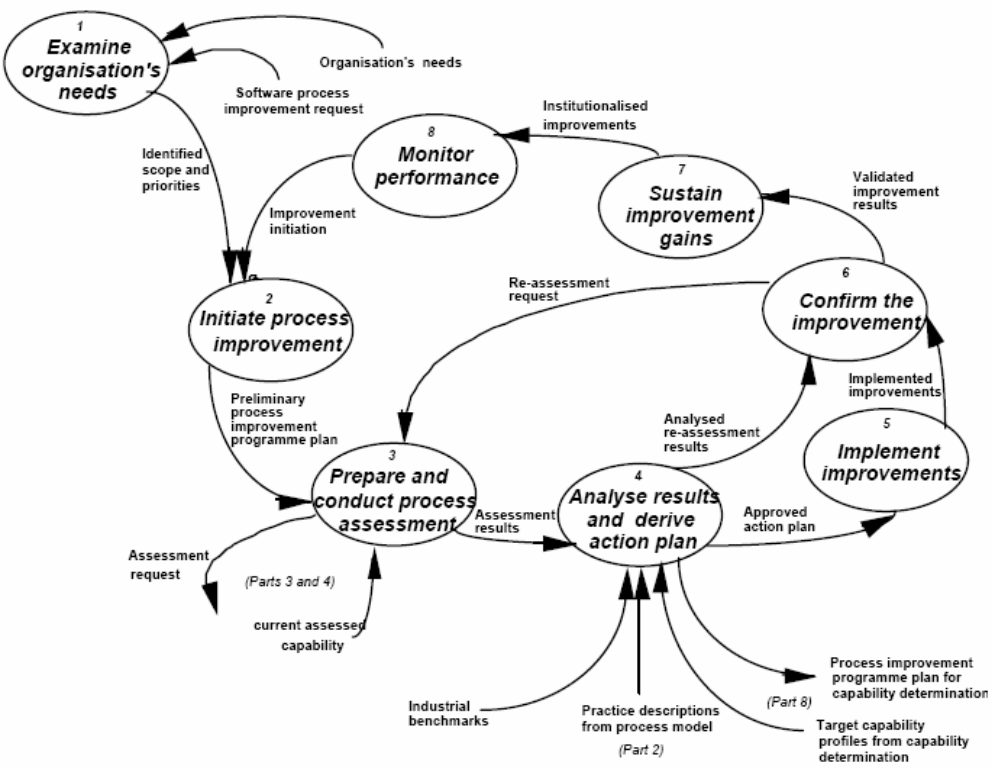

Figure 6.4 Software process improvement steps [18]

In this guide, two alternative approaches to process capability determination are presented. Core process capability determination is a minimum, streamlined set of activities applicable whenever a single organization needs to identify its current process capability,

without any partners or sub-contractors being involved. Extended process capability determination is applicable when an enhanced capability is needed to be done, or when consortia or sub-contractors are involved. Which case is ever selected, the conduct of process capability determination is described in three separate stages, named:

- Target definition stage
- Response stage
- Verification and Risk Analysis Stage

6.1.9. Vocabulary

Part 9 is a consolidated vocabulary of all terms specifically defined for the purposes of SPICE.

Chapter 7

A Model of Process Assessment for XP@SCRUM

While proposing a process assessment model for XP@SCRUM, ISO/IEC 15504 framework has been a reference guide. The structure of ISO/IEC 15504 has been followed.

7.1. Concepts and Introductory Guide

This process assessment model provides a model for the assessment of XP@SCRUM processes. This model can be used in organizations applying XP@SCRUM or wants to assess their processes in terms of XP@SCRUM.

This model provides a structured approach for the assessment of software processes for the following purposes:

a) by or on behalf of an organization applying XP@SCRUM with the objective of understanding the state of its own processes for process improvement;

b) by or on behalf of an organization applying XP@SCRUM with the objective of determining the suitability of its own processes for a particular requirement or class of requirements;

The model for process assessment:

a) encourages self-assessment;

b) takes into account the context in which the assessed processes operate;

c) produces a set of process ratings (a process profile) rather than a pass/fail result;

The process assessment model is based on assessing a specific process instance. A process instance is a singular instantiation of a process that is uniquely identifiable and about which information can be gathered in a manner that provides repeatable ratings. Each

process instance is characterized by a set of four process capability level ratings, each of which is an aggregation of the practice adequacy ratings that belong to that level.

The model is designed to provide assessment results that are repeatable, objective, comparable within similar contexts, and able to be used for either process improvement or process capability determination in XP@SCRUM.

7.1.1. Assessment Model

Process assessment is an activity that is performed either during a process improvement initiative or as part of a process capability determination exercise.

An assessment is carried out by assessing selected processes against the process model. This two-dimensional model consists of a set of process-specific practices. The assessment output includes a set of process capability level ratings for each process instance assessed.

An assessment is supported by an assessment instrument, or set of instruments, constructed. The process assessment is carried out by an assessor(s).

The requirements of an assessor are listed as below:

- Must have good understanding of Agile Methods including XP and SCRUM
- Must have good understanding of process assessment context
- Effective in verbal and written communication

7.2. A Model for Process Management in XP@SCRUM

This part of XP@SCRUM PCI Model is to document the set of practices fundamental for XP@SCRUM.

This model categorizes XP@SCRUM processes into five process categories.

- Communication
- Planning
- Designing

- Coding
- Testing

This model has four levels for processes.

Level 0, Not Performed

Any process in this level is not performed. This level is very dangerous for company.

Level 1, Performed Partially

In this level, processes are partially performed. Most requirements are missing. This level is can lead to potential risk for the company.

Level 2, Performed Largely

In this level, the processes largely satisfy their purpose. Much effort is spent on these processes.

Level 3, Performed Fully

In this level, processes are performed in a globally satisfying way. In terms of business value, it offers a competitive advantage to company.

7.2.1. Base Practices

In an assessment conducted according to the provisions of this model, the processes included within the scope of the assessment shall be mapped to one or more of the processes defined in this clause.

The assessment shall include all of the base practices of each process within the scope of the assessment.

The five process categories are:

COMM	**Communication**
PLAN	**Planning**
DSGN	**Designing**
CODE	**Coding**
TEST	**Testing**

7.2.2. Communication Process Category

The purpose of the communication is to increase the cooperation, productiveness and to decrease the mistakes.

The input of Communication Process is people and tools and the output of this process is definition/refinement of the communication objectives.

COMM.1 Public status displays

Public status displays informs the customer about the status of the project.

COMM.2 Oral communication

Oral Communication is the fastest way of communication.

COMM.3 On-site customer

Onsite customer is customer representative who sits with the development team. When there is a doubt about implementation, on-site customer explains.

7.2.3. Planning Process Category

The purpose of the planning is to establish an appropriate life cycle model for the project.

The input to this process is Product Backlog and the output of this process is a software life cycle model with descriptions of software activities and tasks to be performed by the project and identification of project controls.

Project plans typically document

– Project purpose and objectives;
– Work products to be developed;
– Software estimates;
– Project risks and mitigation plans;
– Resources allocated to the project activities.

PLAN.1 Build Project Team

The purpose of the “Build project teams” process is to establish project teams with qualified members who can fulfill their responsibilities on their team and work together as a cohesive group.

PLAN.2 Maintain Product Backlog

Product Backlog is a prioritized listing all functionality desired in the final product.

PLAN.3 Sprint Planning Meeting

Customers, users, management, the Product Owner and the Scrum Team determine the next Sprint goal and functionality at the Sprint Planning meeting. The team then devises the individual tasks that must be performed to build the product increment.

PLAN.4 Daily Scrum

Daily Scrum is a daily short meeting to monitor the status of the project.

PLAN.5 Track Progress

To see the status of the project Burndown Chart is used. The Scrum Master should update this chart daily.

PLAN.6 Sprint Review Meeting

The Sprint Review meeting is a four-hour informational meeting. During the meeting, the team presents to management, customers, users, and the Product Owner the product increment that it has built during the Sprint.

PLAN.7 Travel Light

The team must be very fast to response the changes.

7.2.4. Designing Process Category

The purpose of the develop software design process is to establish a software design that effectively accommodates the software requirements; at the top-level this identifies the major software components and refines these into lower level software units which can be coded, compiled, and tested.

The input for *Design Process* is Product Backlog and the output is the architecture of the system.

DSGN.1 Simplicity

XP team develops the software by using the simplest thing that works. This means implementing only the required features. Simple design enables to handle changes that will occur in the future.

DSGN.2 Create System Metaphor

System Metaphor is a vision of what the program is and how the program works.

DSGN.3 Refactoring

Refactoring means changing the source code internally to improve the efficiency, performance and readability of the code.

7.2.5. Coding Process Category

The purpose of coding is to produce executable and independently tested units of software code which implement the components of the software design [18].

The inputs of this process are people, tools and backlog items. And the output is the product.

CODE.1 Establish Coding Standards

The team writes the code on an agreed set of rules. This ensures consistency and simplifies the maintenance process with legible code.

CODE.2 Collective Code Ownership

Collective code ownership means that everybody is responsible for all the code; this, in turn, means that everybody is allowed to change any part of the code [14].

CODE.3 Code Integration

Code integration enables to ensure that everything is going well.

CODE.4 Provide Sustainable Peace

The team members are in it for the long term. They work hard, and at a pace that can be sustained indefinitely. This means that they work overtime when it is effective, and that they normally work in such a way as to maximize productivity week in and week out [42].

CODE.5 Configuration Management

"Configuration Management" is satisfied by using a code versioning tool. The available tools are CVS, VSS, etc.

CODE.6 Paired Development

Paired development increases the communication between the developers, ensures "collective code ownership".

7.2.6. Testing Process Category

The purpose of testing is to ensure that the produced software works as stated in the Product Backlog.

Inputs of *Testing Process* are code and data. And the output of this process is success or fail.

TEST.1 Perform unit tests

Unit tests are automated tests written before code is written. These tests are written to check if the functionality of the written code will pass. Before the release, all unit tests must pass.

TEST.2 Code and fix

When a bug is found, a test must be written for it.

TEST.3 Perform Functional Tests

The functional tests are done by customer. Functional tests enables the working software is the software customer wants.

7.3. Rating Process

A base practice adequacy rating or a base practice existence rating shall be determined and validated for every base practice within each selected process instance for each process and/or extended process identified within the assessment scope [18].

Base practice adequacy shall be rated using the maturity level as defined in 7.2.

7.4. Guide to Process Assessment

Process assessment is undertaken to understand the current process. The process model defines, for each process, a set of base practices essential to good software engineering, and a set of generic practices grouped into capability levels. The assessment output consists of a set of generic practice adequacy ratings and process capability level ratings for each process instance assessed together with the assessment record.

The assessment output identifies the current process capability level ratings of an organizational unit's processes and forms the basis to plan, prepare, implement and evaluate specific improvement actions.

The assessment output allows an organizational unit to identify, analyze and quantify its strengths, weaknesses and risks.

Assessment stages are listed as below:

- reviewing the assessment input
- selecting process instances
- preparing for assessment
- information collecting and verification
- determining the actual rating
- validating the ratings
- presenting assessment output

7.4.1. Reviewing the assessment scope

The assessment input shall be defined before the assessment. At minimum the assessment input shall define:

- the assessment purpose
- the assessment scope
- the assessment constraints
- the identity of assessor and responsibilities
- the definition of extended processes
- the identification of additional information

If the assessment is done for a project, the scope of the assessment and constraints is set by customer and the team. If the assessment is done for the organization, the scope of the assessment and constraints is set by the organization.

The assessor should review the defined assessment purpose, scope and constraints to ensure that they are consistent and that the assessment purpose can be fulfilled [18].

7.4.2. Selecting Process Instances

Before the assessment starts, the assessor(s) select(s) the processes to assess. These processes should be agreed with the customer.

7.4.3. Preparing for Assessment

The team for the assessment is created. The size of the team is dependent on the scope of the assessment. The team leader is responsible for overall the assessment. All assessment team members should have experience in software engineering and one or more should have specific experience in the processes under assessment and in the technologies used to support the processes. The team chooses which tools to use for the assessment.

7.4.4. Information Collecting and Validation

Information has to be collected and validated for each process. The information is collected by

- Interviews
- Questionnaires
- Documentation Reviews
- Observation analysis
- Meeting Notes
- Test Results
- Source Code

7.4.5. Determining the Actual Rating

The rating is determined according to the collected information and the metrics defined as below.

COMM.1 Public status displays

Level 0: The customer cannot see the status

Level 1: The customer sees the status rarely (monthly).

Level 2: The customer sees the status frequently (weekly).

Level 3: The customer sees the status continuously (daily).

COMM.2 Oral communication

Level 0: There is not any oral communication between the team and the team is not working the same room.

Level 1: There is a little oral communication between the team but the communication is not performed in a satisfactory way.

Level 2: The team performed the oral communication mostly in a satisfactory way but there are little problems.

Level 3: Team members can learn what they need from other members immediately by oral communication.

COMM.3 On-site customer

Level 0: There is no on-site customer or the assigned on-site customer cannot be reached.

Level 1: An on-site customer is assigned but it is difficult to contact with him/her. He/she cannot populate backlog and prioritize backlog. He/she does not have a good understanding of the project.

Level 2: An on-site customer is assigned. He/she is mostly available for the developer team. He/she is good at populating backlog and prioritizing backlog. He/she has a good understanding of the project.

Level 3: An on-site customer is assigned. He/she is always available for the developer team. He/she is very good at populating backlog and prioritizing backlog. He/she has an excellent understanding of the project.

PLAN.1 Build Project Team

While assessing this process the following activities are considered.

- Define Project Team(s)

  Usually a Scrum team is 6-10 people. If there are more people, split the people into more than one team.

- Assign Roles

  Assign a role (developer, tester, component developer, etc) for each user and assign a duty (component development, interface coding, test, etc) for each Scrum team and reduce the interactions between Scrum teams.

- Establish working environment

  All the team members should be in the same room. The working environment must increase the communication between the team members.

Level 0: The project team is not defined.

Level 1: The project team is defined but the roles are not assigned or the assigned roles are not clear. The working environment is not suitable for working.

Level 2: The project team is defined and the assigned roles are clear but the team(s) is (are) not enough for the project. The established working environment is suitable for working.

Level 3: The project team is defined. The assigned roles are clear. The defined team(s) is (are) enough for the project. The established working environment is very good and motivates the team(s).

PLAN.2 Maintain Product Backlog

While assessing “Maintain Product Backlog”, the following activities are considered.

- Get Customer Requests

  The Product Backlog is driven by the customer. The customer populates the list and prioritizes the list.

- Make Estimation

  After Backlog is created, the team makes an estimation how long it will take. The estimation is done by talking to the people who understands the product and the technology. The estimate includes the time it takes to perform all of the requisite architecture, design, construction and testing.

Level 0: The product backlog does not exist.

Level 1: The product backlog exists but the customer is not good at populating and prioritizing the product backlog. The team is far from making good estimation.

Level 2: The product backlog exists but the level of populating, prioritizing backlog by customer and making estimation by the team is average.

Level 3: The product backlog exists. The customer is good at populating and prioritizing the product backlog. The team is makes good estimation.

PLAN.3 Sprint Planning Meeting

While assessing “Sprint Planning Meeting”, the following activities are considered.

- Define Sprint Goal

  The Product Owner chooses the top priority items and a Sprint Goal is defined based on this priorities.

- Detail the tasks

  The team determines the work to be performed to reach to the Sprint Goal. The tasks must be detailed enough to turn the Product Backlog into a working software. This task is called Sprint Backlog.

Level 0: The meeting is not held.

Level 1: The meeting is held but the sprint goal is not defined and the tasks are not detailed.

Level 2: The meeting is held. Sprint goal is defined but the tasks are not detailed.

Level 3: The meeting is held. Sprint goal is defined and the tasks are detailed.

PLAN.4 Daily Scrum

While assessing "Daily Scrum", the following activities are considered.

- Make short meeting

  The length of this meeting is 15 – 30 minutes. This meeting should not be a long meeting.

- Only ask three questions

  Only the three questions are asked in this meeting.

  - What have you done since the last Scrum?
  - What will you do between now and the next Scrum?
  - What got in your way of doing work?

- Do not turn into design session

Level 0: The daily scrum meetings are not held.

Level 1: The daily scrum is held (daily or frequently) but the meeting does not reach to its aim. The three questions are not asked or other questions are asked. The meeting takes more than 15 -30 minutes.

Level 2: The daily scrum is held (daily or frequently): The meeting mostly satisfies the aim of daily scrum.

Level 3: The daily scrum is always held daily. The meeting reaches to its aim. Only the three questions are asked. It shows the status of the project. The meeting is short.

PLAN.5 Track Progress

Level 0: The burndown chart is not used.

Level 1: The burndown chart is used but it updated rarely.

Level 2: The burndown chart is used but it is updated frequently.

Level 3: The burndown chart is used and it is updated continuously.

PLAN.6 Sprint Review Meeting

Level 0: The meeting is not held at the end of the sprint.

Level 1: The meeting is held but the aim of the meeting is not reached.

Level 2: The meeting is held. The aim of this meeting is mostly reached.

Level 3: The meeting is held and the aim of this meeting is reached. The sprint goal is reviewed.

PLAN.7 Travel Light

While assessing “Travel Light”, the following activities are considered.

- Reduce Documentation

  The team writes only required documentation. This does not mean writing no document.

- Discard temporary models

  Models exist mainly for communication and understanding; discard these once they have served their purpose.

- Plan when needed

  Do not make unnecessary plans.

Level 0: The team is not traveling light. The team does not perform the required actions to perform light.
Level 1: The team starts to understand the importance of traveling light and makes a little effort to travel light.
Level 2: The team has a good understanding of traveling light and makes much effort to perform this.
Level 3: The team always travels light and performs only the necessary actions.

DSGN.1 Simplicity

Level 0: The simplicity is not concerned.
Level 1: The simplicity has a low priority. The system and product is complex.
Level 2: The simplicity has a high priority but there is little complexity in the system and product.
Level 3: The simplicity has the first priority and there is no complexity.

DSGN.2 Create System Metaphor

Level 0: The metaphor is not created.
Level 1: The metaphor is created but it is weak and cannot be understood by many members.

Level 2: The metaphor is created and understood by many members. But there is a little weakness in the metaphor.
Level 3: The metaphor is created and understood by every member. It exactly tells what is wanted.

DSGN.3 Refactoring

Level 0: No refactoring is performed by the developers.
Level 1: Refactoring is performed rarely and a tool is not used.
Level 2: A tool is used for refactoring and refactoring is performed rarely.
Level 3: A tool is used for refactoring and refactoring is performed frequently.

TEST.1 Perform unit tests

The rating level is given according to the followings:

- First test then code
- All code must have unit test
- All tests must succeed before the release
- All unit test must be automated

Level 0: There is no unit test.
Level 1: The unit tests are written and succeed for a few part (< %40) of the code.
Level 2: The unit tests are written and succeed for most part (%40 - %99) of the code.
Level 3: All code has a unit test. All tests succeed before the release.

TEST.2 Code and fix

Level 0: The code is not fixed when a bug is found.
Level 1: The code is fixed but a test is not written.
Level 2: The code is fixed and a test is written for that bug mostly.
Level 3: The code is fixed and a test is written for that bug always.

TEST.3 Perform Functional Tests

The rating level is given according to the followings:

- Customer writes functional tests
- Must be sufficient to test the requirements

Level 0: The functional tests are not performed.

Level 1: The functional tests are performed but they test a few part of the requirements.

Level 2: The functional tests are performed and they test most part of the requirements.

Level 3: The functional tests are performed and they test all of the requirements.

CODE.1 Establish Coding Standards

Level 0: The coding standard is not established.

Level 1: The coding standard is established but the developers do not concern about these rules.

Level 2: The coding standard is established and the developers try to obey these rules but not all of the code satisfies these rules.

Level 3: The coding standard is established and the developers try to obey these rules. All of the code satisfies these rules.

CODE.2 Collective Code Ownership

Level 0: Every developer only knows his/her code and cannot modify others' code.

Level 1: Some of the team members knows about others' code but cannot modify.

Level 2: Many of the team members own others' code and can modify.

Level 3: Every team member owns all the code and can modify any code.

CODE.3 Code Integration

Level 0: There is not a tool for code integration.

Level 1: There is a tool for code integration but it is used before the release.

Level 2: There is a tool for code integration but the code in integrated infrequently (interval is more than one day).

Level 3: There is a tool and there are nightly builds.

CODE.4 Provide Sustainable Peace

Level 0: The team makes overtime frequently and this is not taken into consideration.

Level 1: This is taken into consideration but the team still makes overtime because of poor management.

Level 2: The team makes overtime infrequently.

Level 3: There is no overtime. Everything is going well.

CODE.5 Configuration Management

Level 0: A code versioning tool is not defined.

Level 1: A code versioning tool is defined but it is not used.

Level 2: A code versioning tool is defined but it is rarely.

Level 3: A code versioning tool is defined but it is completely.

CODE.6 Paired Development

Level 0: There is no paired development.

Level 1: The paired development is performed when a problem occurs.

Level 2: The developers try to perform the paired development.

Level 3: The developers perform the paired development more than two hours a day. Everyone changes his/her partner.

7.4.6. Validating Ratings

The ratings should be validated to ensure that they are an accurate representation of the processes assessed. The validation should include assessing whether the sample size chosen is representative of the processes assessed and that it is capable of fulfilling the assessment purpose.

The following mechanisms are useful in supporting validation:

- comparing results to those from previous assessments for the same organizational unit;
- looking for consistencies between connected or related processes;
- looking for proportional ratings across the capability levels e.g. higher ratings for higher levels than for lower ones;
- taking an independent sample of ratings and comparing them to the assessment team ratings;
- feedback sessions of preliminary findings to the organizational unit.

7.4.7. Presenting Assessment Output

After conducting the assessment, the assessment output is needed to be prepared.

The assessment output should contain:

- the assessment input
- the assessment instrument used
- the base practice ratings for each process instance assessed
- the date of assessment
- the name of assessor
- additional information
- the project and sprint name
- any assessment assumption and limitations

The assessment output is a basis for process improving or capability determination. The result of the assessment output is two dimensional chart, process versus rating.

The result data is reported everyone involved in the project including customers.

7.5. Comparison of ISO SPICE and XP@SCRUM PCI

The following table shows the comparison of ISO SPICE (the reference guide) and XP@SCRUM PCI.

Table 7.1 Comparison table of ISO SPICE and XP@SCRUM PCI

ISO SPICE	XP@SCRUM PCI
Contains 9 parts	Contains 4 sections
Six maturity levels Level 0 – Not performed Level 1 – Performed informally Level 2 – Planned and tracked Level 3 – Well defined Level 4 – Quantitatively controlled Level 5 – Continuously improving	Four levels Level 0 – Not performed Level 1 – Performed Partially Level 2 – Performed Largely Level 3 – Performed Fully
Five Process Categories Customer-Supplier Engineering Project Support Organization	Five Process Categories Communication Planning Designing Coding Testing
Self or Independent Assessment	Self Assessment

Chapter 8

A Tool for XP@SCRUM PCI

Another aim of this thesis is to create a tool for the proposed model. The tool is designed for one company and multiple users.

There are three modules in this tool.

- Assessment
- User Management
- Company Management

8.1. Assessment

There are three actions in this module.

- List Assessments (Figure 8.1)
- New Assessment (Figure 8.2)
- Update Assessment (Figure 8.3)
- Delete Assessment
- Assessment Report (Figure 8.5)

When a user logs into the system, the available assessments are listed Figure 8.1.

The properties of assessment are: (Figure 8.4)

- Rating Level
- Evidence&Findings

After rating the processes, a chart is generated like in the Figure 8.5. Also an assessment report is generated Figure 8.6.

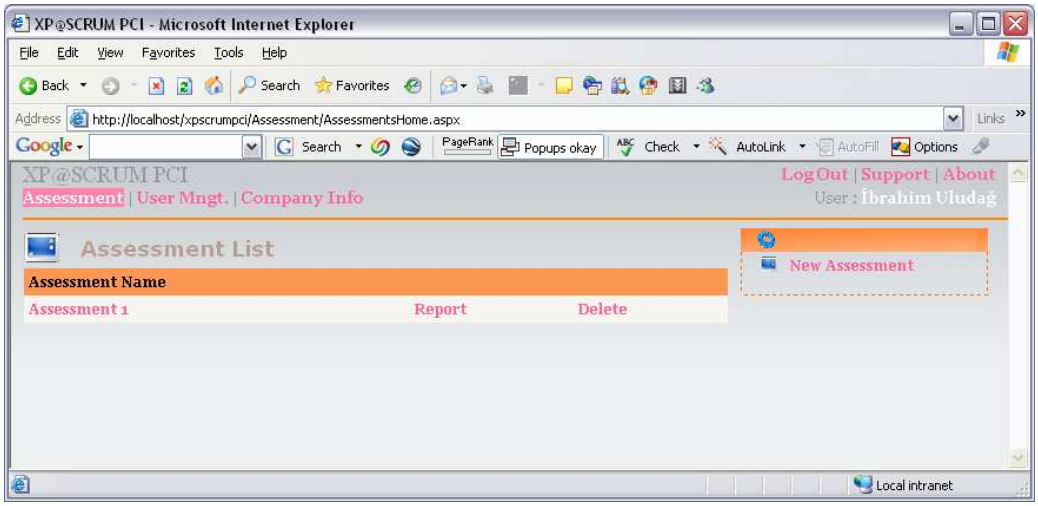

Figure 8.1 Project List

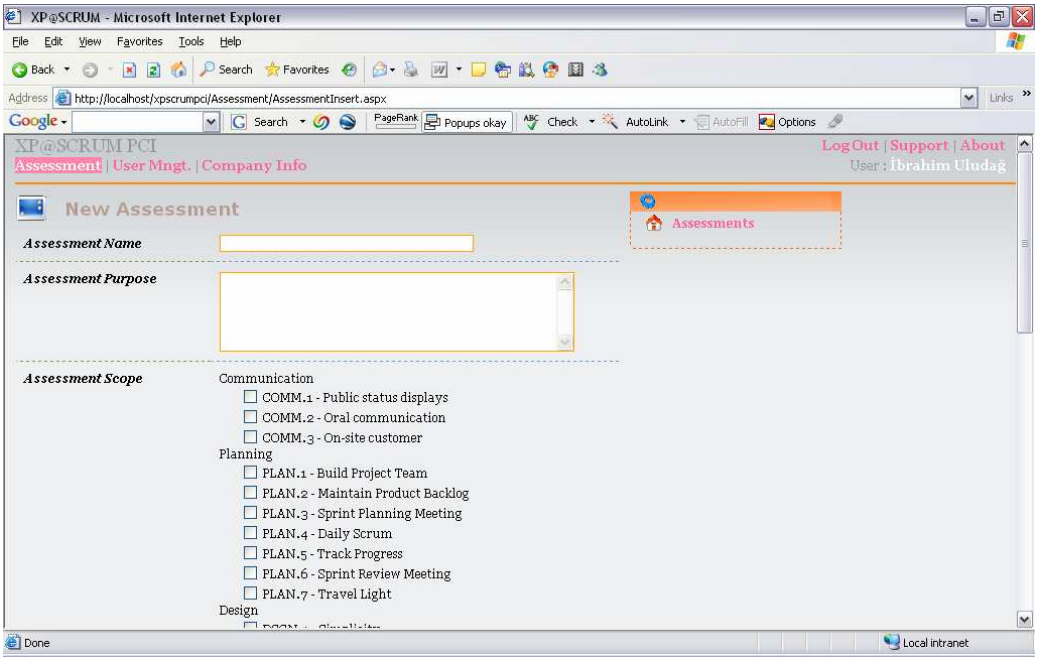

Figure 8.2 Add a new assessment

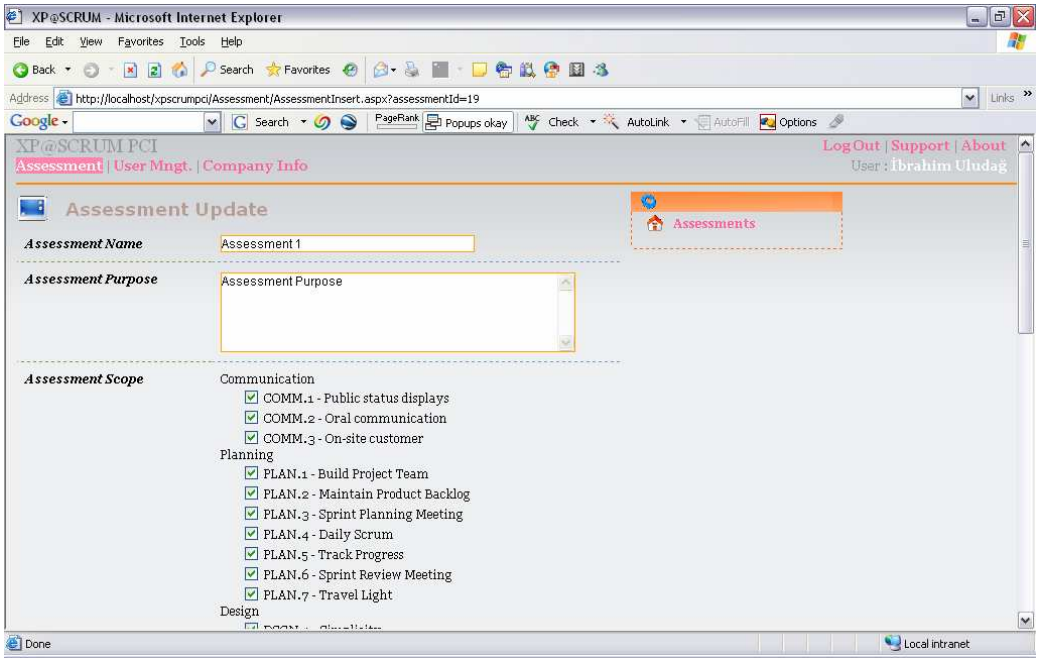

Figure 8.3 Update Assessment

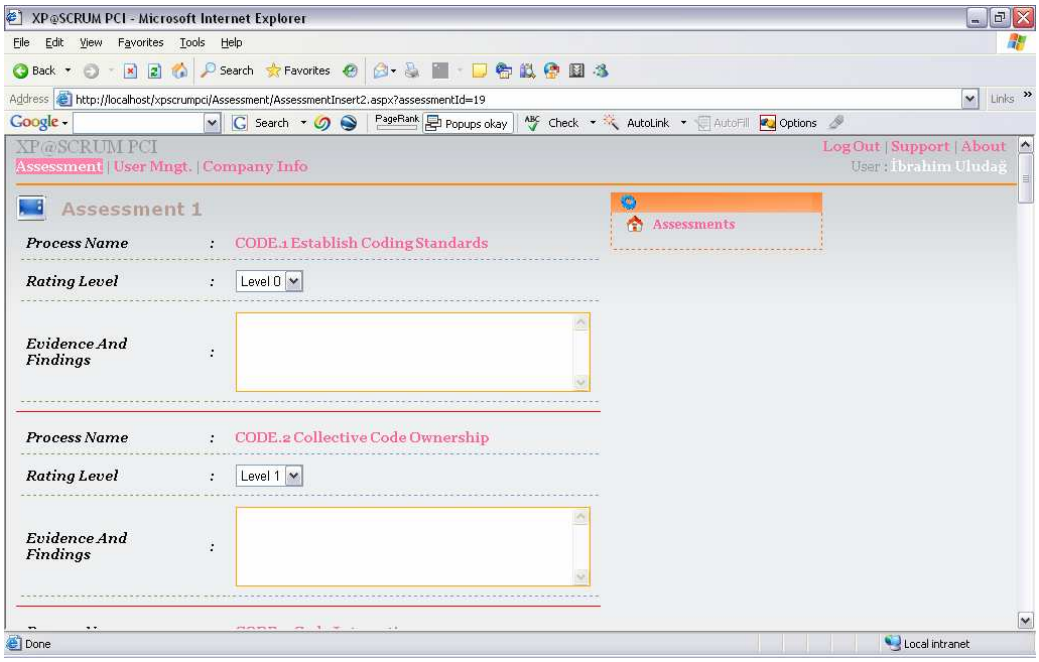

Figure 8.4 Assessment of a Process

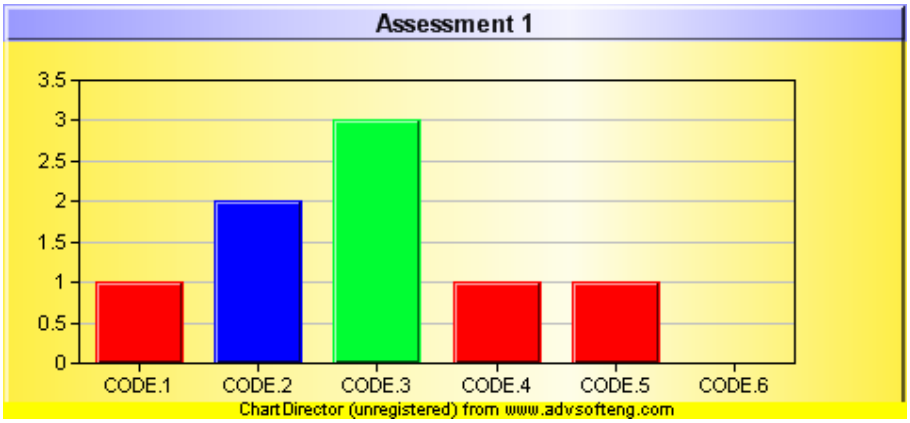

Figure 8.5 Assessment Result Chart

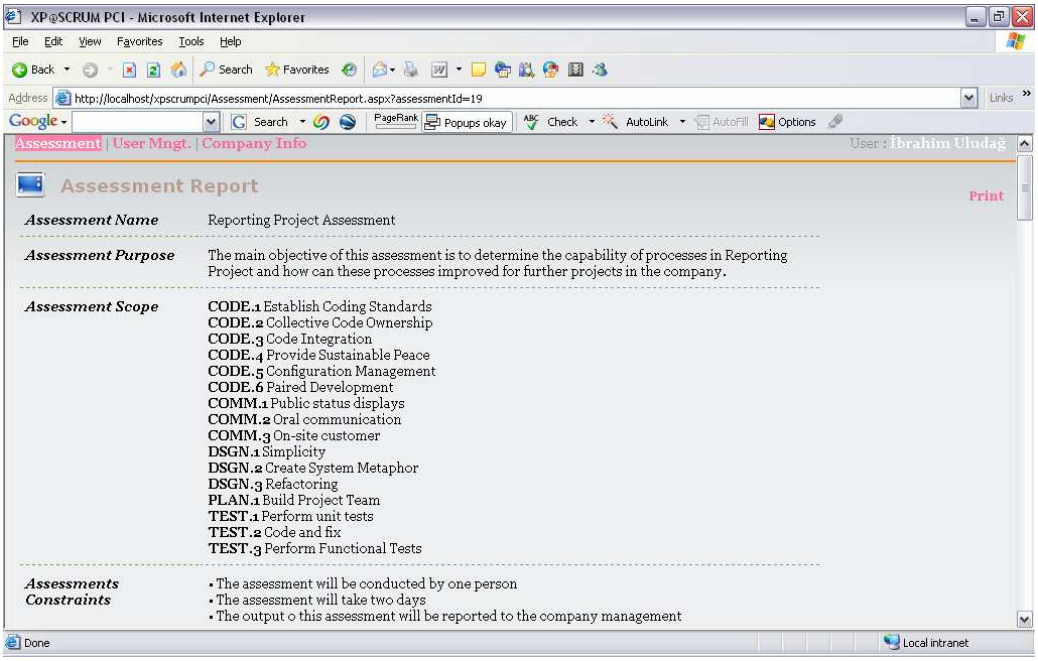

Figure 8.6 A Sample Assessment Report

8.2. Technical Details

This tool is a web based tool. It is created by using Visual Studio Express and MS SQL Server 2005.

The requirements for the server are:

- ASP.NET 2.0 Framework

- IIS 5.0 or higher
- MS SQL Server 2005
- 50 MB free disk
- 512 MB RAM

The requirements for the clients are:

- Internet Explorer 6.0 or compatible browser.

Chapter 9

An Example Assessment

In this chapter, a project will be assessed according to this model. The company is applying agile methodology but the applied method is not XP@SCRUM. So the necessary changes were done in the scope.

9.1. Assessment Scope

Assessment Purpose: The main objective of this assessment is to determine the capability of processes in Reporting Project and how can these processes improved for further projects in the company.

Assessment Scope: The processes below will be assessed in the assessment.

COMM.1 Public status displays
COMM.2 Oral communication
COMM.3 On-site customer
PLAN.1 Build Project Team
DSGN.1 Simplicity
DSGN.2 Create System Metaphor
DSGN.3 Refactoring
CODE.1 Establish Coding Standards
CODE.2 Collective Code Ownership
CODE.3 Code Integration
CODE.4 Provide Sustainable Peace
CODE.5 Configuration Management
CODE.6 Paired Development
TEST.1 Perform unit tests
TEST.2 Code and fix
TEST.3 Perform Functional Tests

Assessment constraints:

- The assessment will be conducted by one person
- The assessment will take two days
- The output o this assessment will be reported to the company management

9.2. Preparing for Assessment

The assessment will be conducted by project leader. The tool created for this model will be used.

9.3. Information Collecting and Validation

The information is collected by means of

- Documentation Reviews
- Observation analysis
- Meeting Notes
- Source Code
- Email messages
- Questionnaires

The collected information is validated by developer team and customer.

9.4. Determining the Actual Rating

The ratings of assessed processes are listed as below:

COMM.1 Public status displays

Rating Result: Level 1

Evidence & Findings: The project has been divided into milestones. The customer could see the status only at these milestones. This is bad because the customer could not see the status of the project continuously.

COMM.2 Oral communication

Rating Result: Level 3

Evidence & Findings: The communication between the company and customer occurred in oral way and these conversations were written by means of email. The communication between the developer team occurred in oral way and emails.

COMM.3 On-site customer

Rating Result: Level 1

Evidence & Findings: There was a customer representative but the customer representative was not dedicated. But he was available at the phone and two meeting was held with the customer.

PLAN.1 Build Project Team

Rating Result: Level 3

Evidence & Findings: Before the project starts, the developer team with responsibilities and the customer representative were assigned.

DSGN.1 Simplicity

Rating Result: Level 1

Evidence & Findings: The project is not the simplest solutions because of the complexity of the requirements.

DSGN.2 Create System Metaphor

Rating Result: Level 3

Evidence & Findings: The developer team had a vision how the final software will be. The metaphor was created by the system requirements document, meeting and phone conversations.

DSGN.3 Refactoring

Rating Result: Level 1

Evidence & Findings: The refactoring was done only when the performance of a produced report is low.

CODE.1 Establish Coding Standards

Rating Result: Level 1

Evidence & Findings: The coding standard defined by Microsoft is used. The code is validated by a tool FxCop. The report tells to make corrections in the code.

CODE.2 Collective Code Ownership

Rating Result: Level 0

Evidence & Findings: Every developer knows only his code. So there is not collective code ownership.

CODE.3 Code Integration

Rating Result: Level 1

Evidence & Findings: The code is integrated only before milestones.

CODE.4 Provide Sustainable Peace

Rating Result: Level 0

Evidence & Findings: The developers worked more than 40 hours a week.

CODE.5 Configuration Management

Rating Result: Level 2

Evidence & Findings: The CVS tool used for version controlling for the code. But there is no version controlling tool for database is used.

CODE.6 Paired Development

Rating Result: Level 0

Evidence & Findings: There is no paired development.

TEST.1 Perform unit tests

Rating Result: Level 0

Evidence & Findings: There are no automated unit tests.

TEST.2 Code and fix

Rating Result: Level 0

Evidence & Findings: There is not code and fix method.

TEST.3 Perform Functional Tests

Rating Result: Level 1

Evidence & Findings: The customer performed functional tests after each milestone. But the functional tests were not satisfactory because some bugs could not be found by the customer.

9.5. Presenting Assessment Output

The two dimensional chart of the result of the assessment is shown in the Figure 9.1.

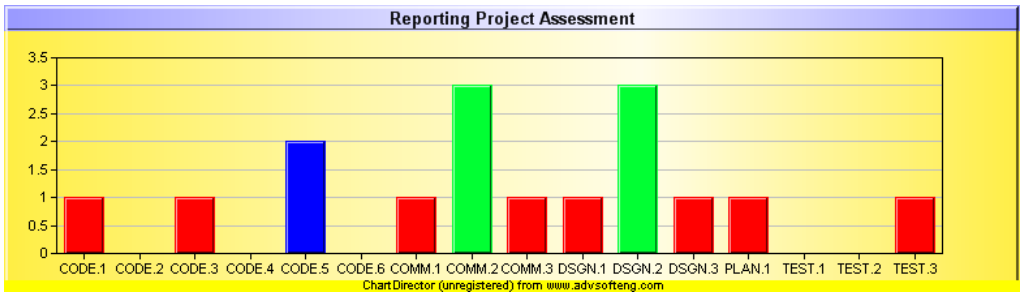

Figure 9.1 Assessment Result of Reporting Project

The result of the assessment says that the company shall make improvements in many processes. The weakest part of the organization is Testing Process Category.

The result of the assessment was presented to the company management and the management decided to hire a tester to improve the testing process category.

Chapter 10

Conclusion

In this thesis, a lightweight software process assessment model and a tool for XP@SCRUM called XP@SCRUM PCI has been proposed. ISO/IEC 15504 has been a reference process assessment framework.

The companies using XP@SCRUM or other agile methods can easily adapt this model as process assessment model, because this model is a light process assessment model. There is no need for a trained assessor. This model uses the organizations' resources. Usually the Scrum Masters take the role of an assessor. The process categories and the processes are specific to this agile method. But by customizing the process categories and processes, this model can be used with other agile methods.

By applying this assessment method and tool, the companies will increase their ROI and customer satisfaction. Organizations can determine the strong and weak processes. After determining the weak processes, the organizations can improve the weak processes.

This model is between "No Assessment" and "International Standards". After reaching Level – 3 for all processes, the company may leave using this model and apply an International Standard. The possible International Standards are ISO/IEC 15504 and CMMI. The companies can get a CMMI or ISO/IEC 15504 level with a little effort after reaching Level – 3.

References

[1] About Scrum – Overview, http://www.controlchaos.com/about

[2] Abrahamson P., Salo, O., Ronkainen, H., & Warsta J., (2002). Agile Software Development Methods Review and Analysis, University of Oulu

[3] ADM Inc., "xP@Scrum", http://www.controlchaos.com/xpScrum.php

[4] Ambler, S.W. (2002b). Modeling Style Home Page, http://www.agilemodeling.com/style

[5] Andersson. C., & Runeson, P., (2002). "Verification and Validation in Industry: A Qualitative Survey on the State of Practice", Proceedings of the International Symposium on Empirical Software Engineering, Nara, Japan

[6] Beck, K. (1999). Extreme Programming Explained (1st Edition), New York: Addison Wesley

[7] Beck, K. (2000). Extreme Programming Explained: Embrace Change (2nd Edition). New York: Addison-Wesley.

[8] Boehm. B. (2002). Get ready For The Agile Methods, With Care, Computer 35 (1): 64-69

[8] Boehm, B., & Turner R., (2003). Balancing Agility and Discipline: A Guide for the Perplexed: New York: Addison Wesley

[10] Cockburn A., & Williams L. (2000). The Costs and Benefits of Pair Programming, University of Utah, Utah, USA

[11] Cockburn, A. & Highsmith, J., (Sept. 2001) "Agile Software Development: The Business of Innovation," IEEE Computer, pp. 120-122.

[12] Crnkovic, I., Larsson, M., Luders., F., (January 2000). "State of the practice: Component-based software engineering course." In Proceedings of 3rd International Workshop of Component-Based Software Engineering. IEEE Computer Society

[13] Extreme Programming, http://en.wikipedia.org/wiki/Extreme_Programming

[14] Extreme Programming Practices, http://en.wikipedia.org/wiki/Extreme_Programming_Practices

[15] Fowler M. Continuous Integration, http://www.martinfowler.com/articles/continuousIntegration.html

[16] Fowler, M., (June 1999) Refactoring – Improving the Design of Existing Code, Addison-Wesley, ISBN 0201485672

[17] Karlström, D., (November 2002). Increasing Involvement in Software Process Improvement, Lic. Thesis, Technical Report 150

[18] ISO/IEC TR15504:1998 (E), Information Technology, Software Process Assessment Parts 1 to 9

[19] Karlström, D., (2004). Integrating Management and Engineering Processes in Software Product Development, Lund University Department of Communication Systems Lund Institute of Technology

[20] Liu, S., (1998). "SOFL: A Formal Engineering Methodology for Industrial Applications", IEEE Transactions on Software Engineering, Vol. 24, No. 1, pp. 24-45

[21] Mar, K., & Schwaber, K., Scrum with XP, http://www.informit.com/articles/article.asp?p=26057&rl=1

[22] Marchesi, M., & Succi, G. (2001). Extreme Programming Examined (1st Edition). New York: Addison Wesley

[23] Meloche, T. (2002), How to build a world class delivery organization. Michigan, The Menlo Institute

[24] Menon, N. (2006). SCRUM: Saving Projects from Failing, http://www.stylusinc.com/website/scrum.htm

[25] Miller. G.G. (2001). The Characteristics of Agile Software Process. The 39th International Conference of Object-Oriented Languages and Systems (TOOLS 39), Santa Barbara, CA

[26] Nawrocki, J., Walter B., & Poznan, W.W. (2001). Toward Maturity Model for eXtreme Programming, University of Technology, 60-965 Poznan, Poland

[27] Paulk M.C. (2001), Extreme Programming from a CMM Perspective, Pittsburgh: Software Engineering Institute

[28] Principles behind the Agile Manifesto. http://agilemanifesto.org/principles.html

[29] PTY LTD (2006), Scrum Reference Card , ww.contrado.com.au/images/src.pdf

[30] Pyhäjärvi M., (31st November 2004). Seminar on Quality Models for Software Engineering Department of Computer Science, University of Helsinki, Helsinki

[31] Royce,W., W., (August 1970). "Managing the development of large software systems: concepts and techniques", Proceedings IEEE WESTCON

[32] Schokking A. (2005), Project Planning and Tracking System, Competitive analysis, Eindhoven

[33] Schwaber K., & Beedle M., (2002). Agile Software Development with Scrum(2nd Edition) , New Jersey: Prentice Hall

[44] Schwaber, K. Scrum Development Process, http://jeffsutherland.com/ oopsla/schwapub.pdf

[34] Schwaber, K. (2004), Project Management with Scrum (1st Edition), Washington: Microsoft Press

[36] Sim, S.E., Easterbrook, S., & Holt, R.C. (2003). “Using Benchmarking to Advance Research: A Challenge to Software Engineering," in International Conference on Software Engineering. Portland: IEEE, pp. 74-83

[37] Software Quality Institute. Software Process Improvement and Capability Determination. http://www.sqi.gu.edu.au/spice

[38] Sommerville, I. (2001). Software Engineering (6th Edition), New York: Addison-Wesley

[39] Visconti, M., & Curtis R., (2004). An Ideal Process Model for Agile Methods, Oregon State University, Oregon, USA

[40] Vriens, C. (2003). Certifying for CMM Level 2 and ISO9001 with XP@Scrum, Philips Research - Software Engineering Services (SES)

[41] Wells, D., & Williams, L., (2002). “Extreme Programming and Agile Methods -- XP/Agile Universe 2002," in Lecture Notes in Computer Science. Berlin: Springer-Verlag

[42] What is Extreme Programming?, http://www.xprogramming.com/xpmag/whatisxp.htm

[43] Williams, L., Layman, L., & Krebs, W. (2004). Extreme Programming Evaluation Framework for Object-Oriented Languages Version 1.4

[44] XP Flow Chart, http://www.extremeprogramming.org/map/project.html