



Agile Software Development of Embedded Systems

Version : 1.0
Date : 2005.03.24
Pages : 39

Authors

Tuomo Kähkönen

Status

Final

Confidentiality

Public

Agile Deliverable D.2.1

Framework for Agile Software Development in Embedded Systems

Abstract

The purpose of this document is to describe the framework for agile software development developed in ITEA AGILE project. The goal of this document is to define and communicate:

- Terminology used in the framework
- The framework structure
- Definition of the elements in the framework
- Notation used in the framework
- Management of the framework



I T E A

INFORMATION TECHNOLOGY

FOR EUROPEAN ADVANCEMENT

CHANGE LOG

Vers.	Date	Author	Description
0.1	16.12.04	Tuomo Kähkönen	First draft created
0.2	21.01.05	Tuomo Kähkönen	Comments from Minna Pikkarainen. Concept model added and terminology aligned accordingly. First version of Pattern Catalogue added. Example mapping to CMMI added. List of example methods added. Process modelling symbols added.
0.31	10.02.05	Tuomo Kähkönen	Terminology definitions added.
1.0	24.03.05	Tuomo Kähkönen	

TABLE OF CONTENTS

CHANGE LOG	2
1 Introduction	4
2 Terminology	4
2.1 Framework Related Terminology	4
2.2 Additional Definitions	6
3 Framework Goals	6
4 Framework Overview	7
5 Use of the Agile SW Development Framework	8
6 Structure of the Agile SW Development Framework	9
6.1 Values and principles	9
6.2 Agile SW Development	10
6.3 Agile SW Development Method Deployment.....	10
6.4 Example Methods	10
6.5 Toolbox of Agile Practices	12
6.6 Experiences	15
7 Framework relationship to standards	15
8 Framework Management	16
9 Technical Implementation	16
Appendix A: Agile Practice Description Template	17
1 Name	17
2 Context	17
3 Problem	17
4 Solution	17
5 Implementation Guide	18
5.1 Examples.....	18
5.2 Guidelines	18
5.3 Risks.....	18
5.4 FAQ	18
6 Metadata	18
6.1 Pattern editor.....	18
6.2 Pattern origin and relation to other practices	18
6.3 Pattern classification	18
6.4 References.....	19
6.5 Change log	19
Appendix B: Recommended Practice for Process Modelling	20
Appendix C: Principles of Agile SW Development in Embedded Systems	21
Appendix D: Pattern catalogue	34
Appendix E: Mapping of patterns to CMMI process areas	39

1 Introduction

The purpose of this document is to describe the framework for agile software development developed in ITEA AGILE project. The goal of this document is to define and communicate:

- Terminology used in the framework
- The framework structure
- Definition of the elements in the framework
- Notation used in the framework
- Management of the framework

During development time the actual content of the framework is out of the scope of this document. In this version some illustrative examples are provided as appendix of this document. When the project's new web site is up, the latest version of the actual framework content will be found from the project's web site. In the end of the project, the framework content will be consolidated in a separate document.

Intended audience of this document is the community developing the ITEA Agile framework and providing contents to it. Definitions in this document are considered to be mandatory within the scope of the framework with the exception of example methods. It is recommended that the developers providing the example methods align their terminology and notation as much as possible with the definitions in the framework, but if there is any valid business reason to make exceptions, those are allowed.

2 Terminology

2.1 Framework Related Terminology

This chapter defines the terms used in the framework description.

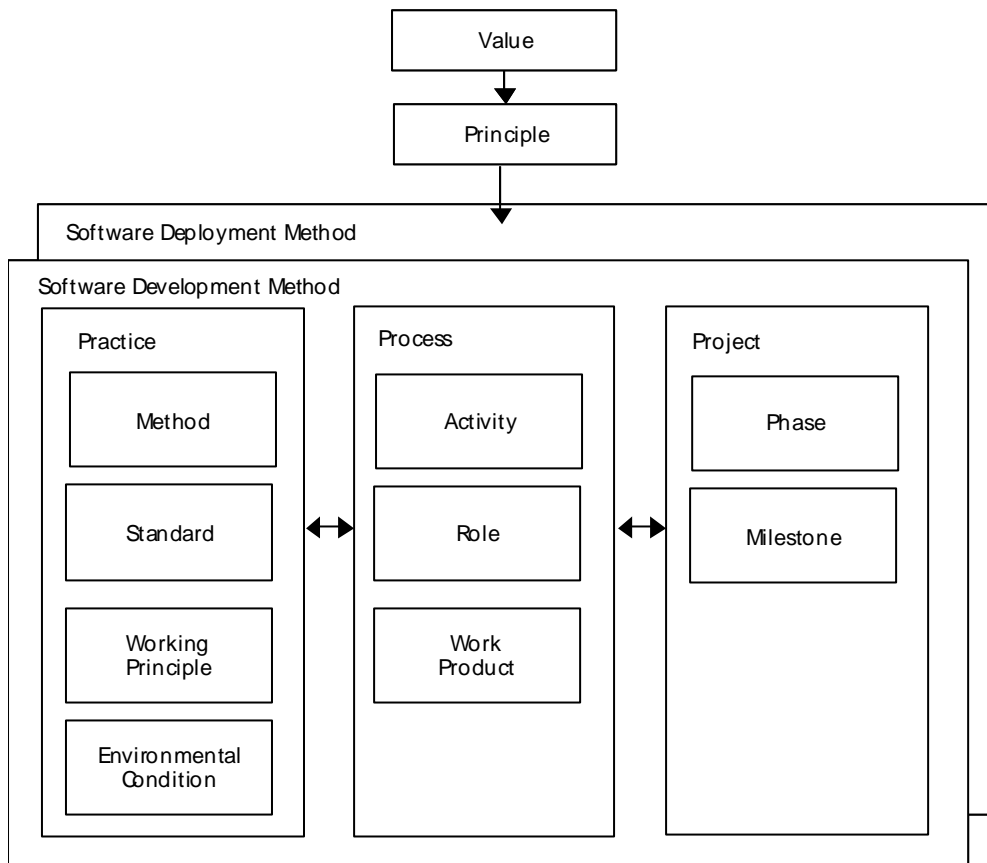


Figure 1: Concept model of the agile development framework

Framework is a logical structure for classifying and organizing complex information. It gives a broad overview, outline or skeleton, within which details can be added. Framework for agile software development is used for organizing different type of information related to the agile software development.

Software development method is the systematic and predefined way the organization works in general to produce software. The software development method may describe organizational behaviour from one or various viewpoints. In traditional software development methods engineering, project management and process aspects have been the most prominent. Agile software development methods have put emphasis on social aspects like values, principles and practices how people collaborate.

Software deployment method describes a systematic way to change the software development method.

Value describes individuals' or organizations' preferences and behavioral priorities. Once individual have internalized organizational values, they become a standard for guiding actions, opinions and attitudes. How an individual accepts organizational values, depends on many factors from which individual's own value system is a large component. Different software development methods have different underlying values. For that reason organization's values restrict what software development methods can be deployed in them.

Principle is a basic generalization that is accepted as true and that can be used as a basis for reasoning or conduct. Principle is a practical guideline that helps in making choices. Principles are derived from the values and they can be in various levels of granularity. They may guide whole organizational behavior or single activity. The later ones are called in this framework working principles.

Practice is development teams collaborative agreement on how to work on some area of process. Practice can be in form of a method, a standard or a working principle.

Method describes systematic and predefined way to carry out a task or activity and is applicable in many different situations. Method is a collection of techniques, steps and rules that give guideline for working. In addition method might describe concepts and notation used in work products, roles, process, objectives, values and assumptions. E.g. planning game, reflection workshop.

Standard describes quality criteria of activities or work products. Standard is a convention that the team adopts for particular tools, work products, decisions and other activities. E.g. coding standard.

Working principle is a practical guideline or a rule that guides team members in their daily routines i.e. when they perform activities or make decisions. E.g. 40-hour week, all green before check-in.

Environmental condition is a prevailing condition or setup that exist continuously. E.g. team in one location, customer always available.

Technique is a prescriptive presentation of how to accomplish a task by using an understood body of knowledge. Some techniques, like writing a use case, apply to a single role and while others are aimed for groups, like reflection workshop.

Process is the way that people work together and apply methods and tools, to produce a predefined result. Thus, processes exist in every organization, whether or not they are explicitly identified and documented. A documented process presents a general model of how its defined outputs are accomplished from the inputs by the organization.

Activity: All the value-adding work of a process is performed within the activities, which transforms specified inputs into predefined outputs. Activities of a process are never isolated. Activities are explicitly connected through their inputs and outputs. Each activity has a number of inputs and preconditions, which are required before the activity can be executed.

Role performs a certain set of activities. A role is responsible for an output to be in a defined state for further processing. A new role needs to be defined e.g. when a competence requirements change between activities or when authority restrictions force a different role.

Work products are the intermediate and final results produced in the activities. Work products may be concrete artefacts, documents, items in information systems or more abstract results (e.g. ASIC chip, requirements definition document, test case, understanding of project status).

Project is a planned undertaking of organized set of activities designed to achieve specific outcomes within a given time frame, resources and budgeted.

Phase: Projects are usually divided into several phases in order to provide better management control. These phases together form the project life cycle. Life cycle also defines the beginning and the end of the project.

Milestone is a significant event in the project, usually completion of a major deliverable. Milestone is usually also a decision point where a steering group decides whether the project should continue or not.

Pattern is a recurring solution for recurring problem. Pattern is a format for defining problem solution pairs and giving contextual information where and how to apply the pattern. In this framework patterns are used for documenting principles and practices.

2.2 Additional Definitions

Methodology term is not used in this framework because it easily causes confusion. Instead methods may contain other methods.

Tailoring refers to activity of making a (project / team etc.) specific version on software development methods from a generic description.

3 Framework Goals

This framework has been designed especially taking into account following challenges:

- Different domains require different process characteristics and project characteristics vary also within organizations. Framework must adapt for those different situations.
- Framework must provide support for selecting appropriate methods and practices based on project characteristics
- Framework must provide practical support for projects taking it in use

- Framework must provide implementation support for the selected methods and practices
- It must be relatively easy to take the framework in use. There should be ready-made methods that can be taken in use – not just individual practices.
- The framework must support method adaptation based on the needs of the target organization
- Framework must adapt on different situations where it is taken in use: from beginning of the project, during the project.
- Framework must support reuse of agile methods and related information
- Framework must help in collecting lessons learned from agile projects
- Framework must provide an well organized overview for agile development practices available
- Framework helps to build a common communication language about agile software development methods and practices

4 Framework Overview

At top level the framework has three parts:

- Values and principles of agile development describing the underlying ideas behind agile methods and practices
- Agile SW Development (= Development) describing how agile software development - including related management and support processes – is done
- Agile SW Development Method Deployment (= Deployment) describing how organization can take agile approach in use and support continuous organizational learning

The development and deployment areas are divided in three parts:

- Example agile SW development and Deployment methods (= Example Methods) are complete packages that can be deployed in organization either for development or deployment purposes. Example methods contain pre-configured sets of the practices from the toolbox. The Example methods may contain also adapted or alternative practises specific for the method. Example methods contain also other elements that are needed in order to make a development or deployment method complete. The method elements vary in the example methods, but they may contain process models, project phase & milestone models etc.
- Toolbox gives generic definitions of individual practices that can be used in development and deployment. Toolbox content is defined as patterns meaning that in addition to the definitions also contextual information is provided. That information helps in selection of the practices and when configuring complete methods from individual related practices. The purpose of the toolbox is to provide modularity in the framework and make it easier to create tailored methods for project / organization specific needs.
- Experiences are a collection of experience reports from the use of methods and individual practices. They may also contain examples on methods that are instantiated for a specific project needs. The format and the content of the experiences may vary.

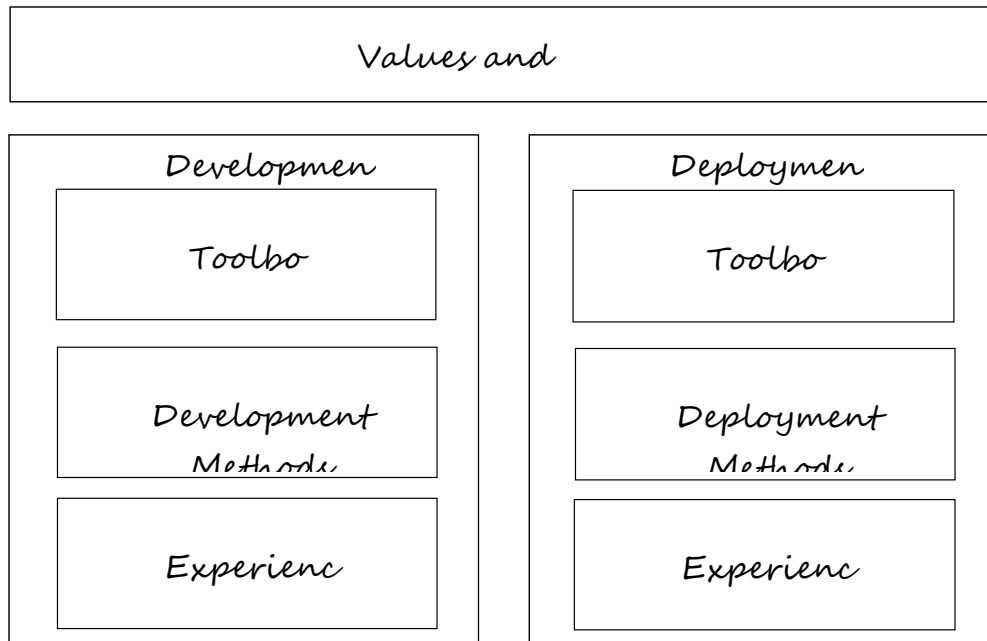


Figure 2: Overview of the framework for agile software development

5 Use of the Agile SW Development Framework

Agile methods can be described at various levels. On one hand there are various *generic agile methods* (XP, SCRUM, Crystal, Mobile-D etc.) that are publicly available and that are not designed for any individual project, organization or a group of organizations.

On the other hand, it may be stated that every project has own method or at least a specific instance of a method. This tailored instance of the method can be called *project specific agile method*. It may be based on a generic agile method, developed from scratch or result from combining existing practices in the organization and some practices from the agile methods.

Organizations may have their own *organization specific methods*. Traditionally those generic organization specific development models are called organization's standard processes. Those organization specific methods are at more general level than project's methods and they serve as templates for projects for developing their own specific methods. This process of elaborating a project specific method from organization specific method is called tailoring in the traditional process improvement literature. In ideal situation the knowledge is transferred in both directions. The organization specific method is refined and modified based on the feedback from the projects that have utilized it. Similar approach can be used also with agile and hybrid methods in order to promote organizational learning.

The framework introduces a new *intra-organizational method* layer that helps the member organizations of the ITEA AGILE consortium to share the best practices of agile software development. The framework consolidates the lessons learned from the generic agile methods and from the specific methods used in the member organizations. The member organizations can benefit from the framework either by applying its content directly to their project specific methods or indirectly by applying its content to their organization specific methods.

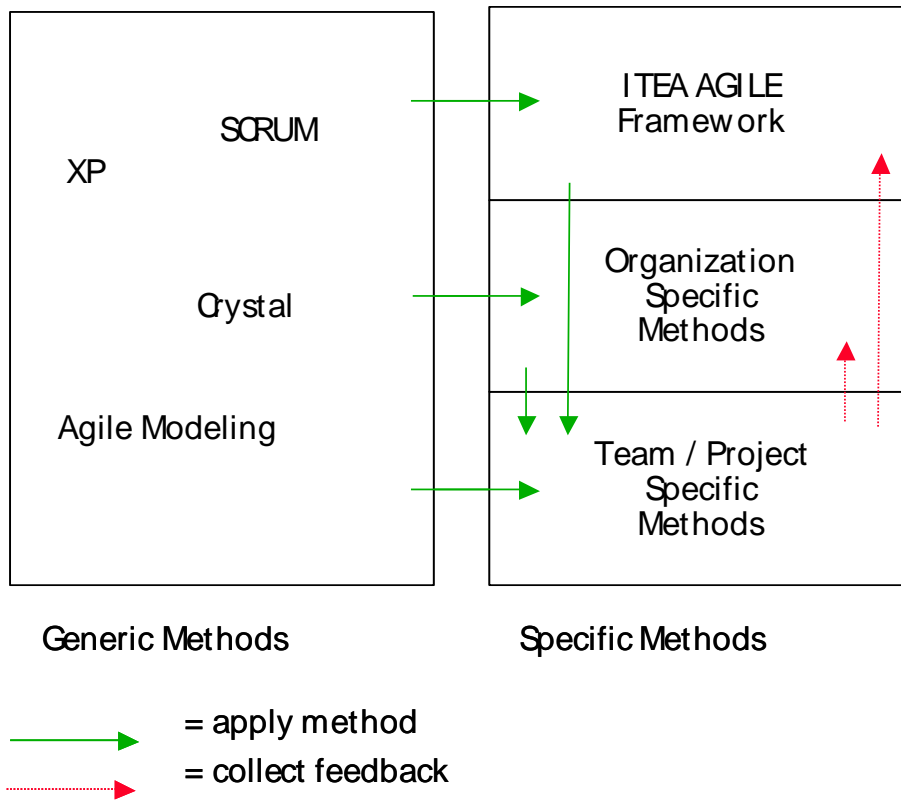


Figure 3: Context of the framework

6 Structure of the Agile SW Development Framework

6.1 Values and principles

Values of Agile Software Development describes the most fundamental building blocks of the mindset of agile software development. Values in general are guiding all human behaviour and thus they have important impact how organization works. By explicitly stating the values it is possible to increase awareness of them and to start gradually changing them to desired direction. However, to change the values is not easy or straightforward and requires lots of time.

Principles of Agile Software Development describes the generic mechanisms to increase agility. As they are in rather abstract level it is not possible to apply them directly in the organization. However, they provide guidance and they are useful when trying to understand why some organizations are able to better accommodate changes than others.

As the values and principles are fundamental elements of agile thinking they affect on every part of the framework. Content of both development and deployment parts of the framework are designed so that they follow the values and principles of agile software development. The values and principles should be used as the main criteria when judging whether some practices or techniques should be included in the framework or not.

Values are described in appendix X. The description is a free-format textual description. Principles are described in appendix X as patterns using the template provided in Appendix X.

6.2 Agile SW Development

Development area of the framework describes how software can be developed agile way. Development should be interpreted broadly including all development, management and support activities that are needed to develop software.

It should be noted that agile toolbox includes only development practices that can be classified as agile whereas example methods may combine agile and more traditional practices to provide a mixture that fits for particular domain or business needs.

6.3 Agile SW Development Method Deployment

Deployment area of the framework describes how agile method can be taken in use in organizations. Deployment covers the whole lifecycle of agile projects including the fit-for-use analysis, deployment planning, initial deployment, continuous improvements and management of the end of the agile development.

Many of the deployment practices presented in the framework are not specific for agile development. However, the agile approach may influence on how the practice or method should be applied. Thus emphasis is put on how the deployment is carried out in agile environment.

6.4 Example Methods

Example methods are coherent collections of practices, processes, activities, roles, work products, life cycles and organizational models that can be applied in organization as one package.

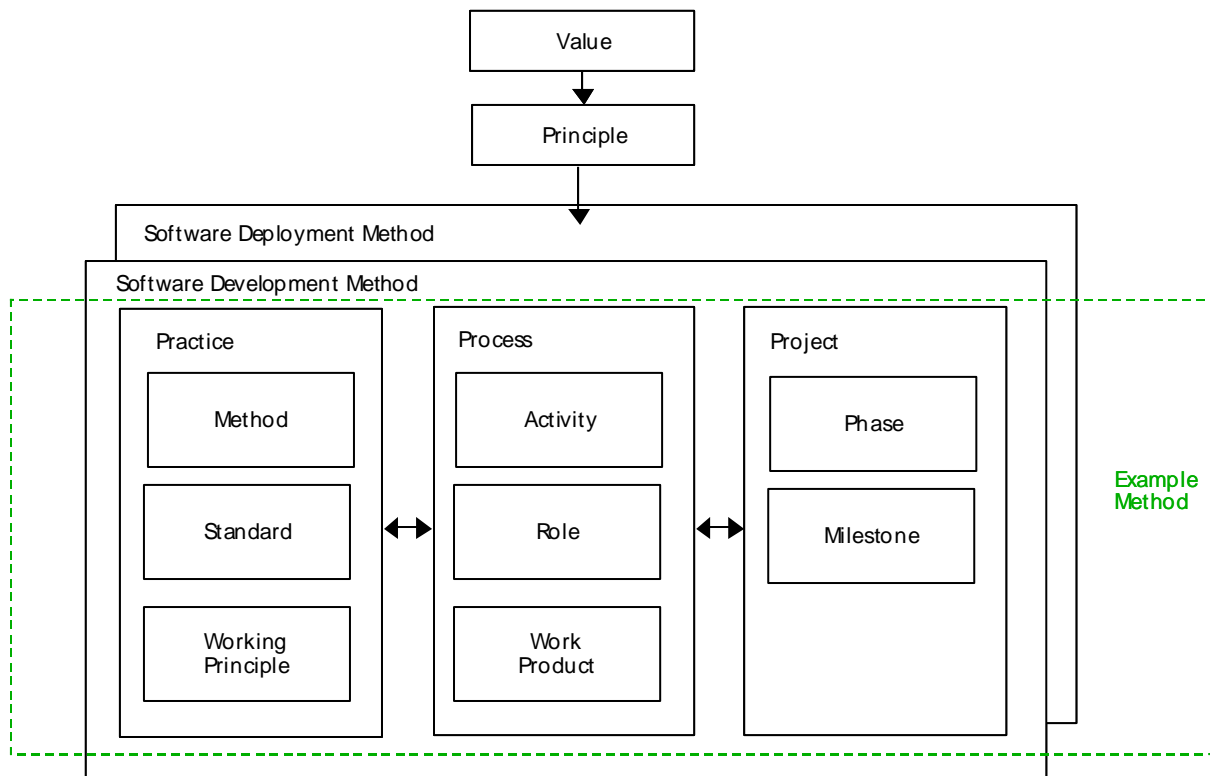


Figure 4: Scope of example methods vs. concept model

The framework contains multiple methods that are designed for different purposes. They can be designed e.g. for a specific domain or a specific type of project. The method should state also the context where it has been intended and the assumptions made when the method was designed. Purpose of this information is to help framework users to select appropriate methods.

Example methods are provided for organizations as a starting point for developing their own method instance. The example methods can be used several ways. The most important approaches are:

- Method can be applied by-the-book so that a project / team makes their own instance of the method as closely to the description as possible.
- The most appropriate method can be used as a starting point when a project / team starts to develop their own method. Practices can be added, removed or modified in order to make the method to fit the needs of the team / project.
- Method is used as a source of ideas when developing project / team / organization specific method.

A list of example methods and references to them is provided in the appendix X.

The developers of the example methods may use any notation that conforms the specific organizational and domain needs. The methods may either refer directly to the patterns in the agile toolbox or include own localized instances of the patterns. The concept model of the framework defines the most commonly used process modelling elements and their definitions. It is recommended to use these terms if possible. Example notation for these modelling techniques is provided in appendix X.

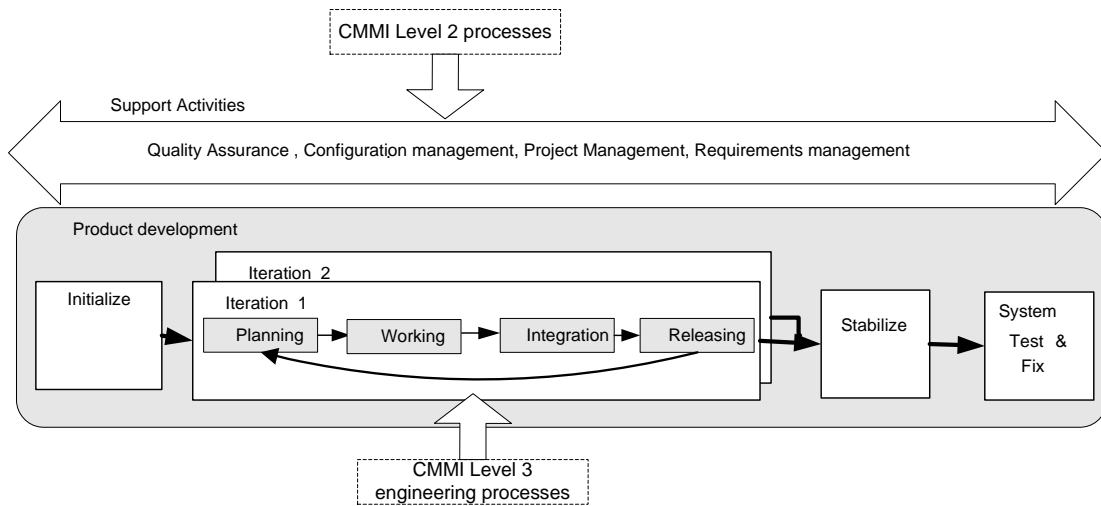


Figure 5: Mobile-D is an example development method

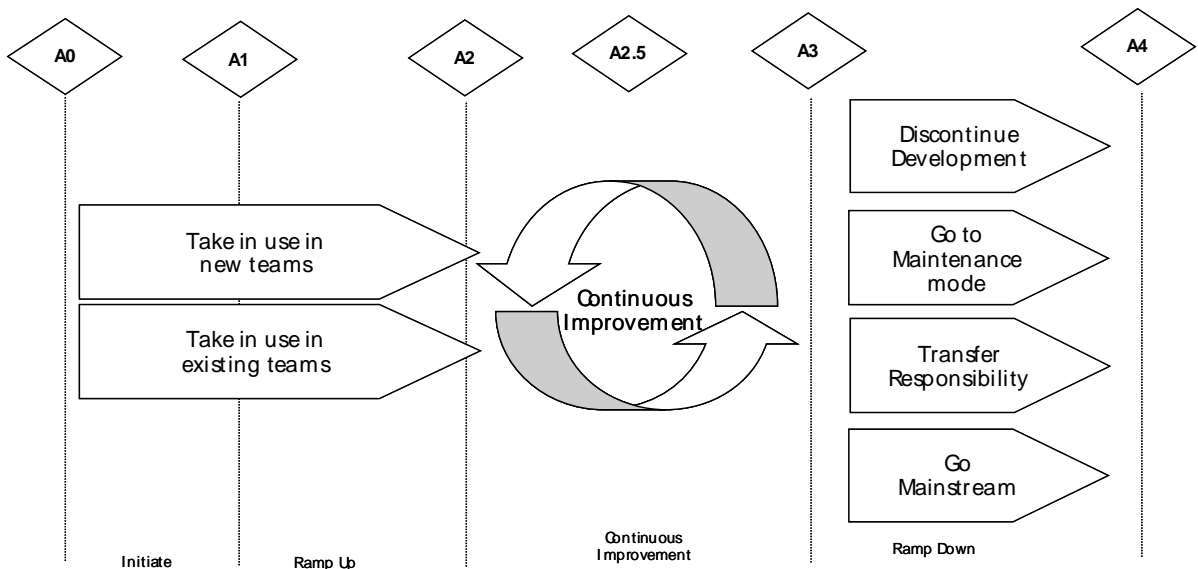


Figure 6: Agile deployment project model is an example deployment method

6.5 Toolbox of Agile Practices

The toolbox of the patterns describes practical ways to take the principles in use. An important difference between patterns and principles is that patterns are context dependable whereas principles are invariants that can hold globally.

The main criteria for which practices are included in toolbox is the conformance to agile values and principles. If a practice follows the agile values and principles and furthers their practical implementation in a certain context, the practice can be included in the toolbox. The starting point of the toolbox is the practices proposed by various generic agile methods but the toolbox is by no means limited to those. Addition of new innovative practices is thus encouraged.

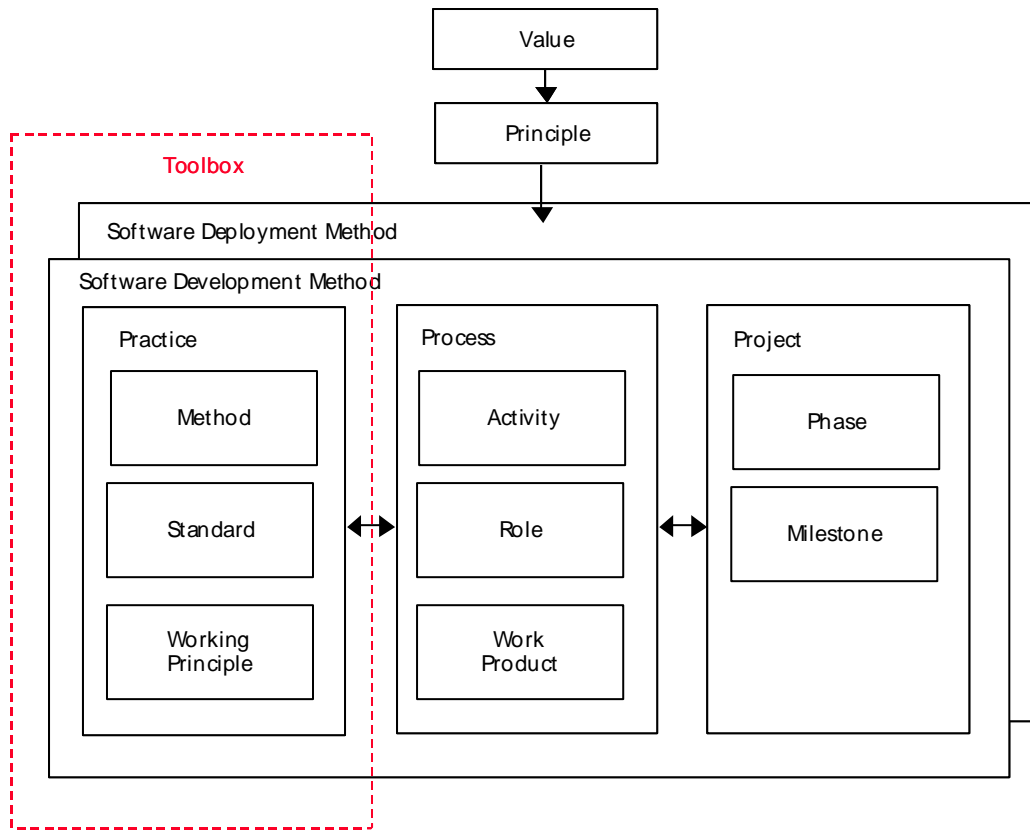


Figure 7: Scope of the toolbox vs. concept model

Practices in the toolbox are defined as patterns. Because the nature of the patterns there is no one right way to define the patterns. Patterns defined in different context may be overlapping or may have different connotations. Patterns also evolve during time. They represent the learning process that happens among the people developing them. As understanding increases, some patterns may be split, combined or a new pattern emerges from the existing ones. This nature of patterns puts some challenges to their management. The goal of the toolbox is to have one set of patterns that have as little overlap as possible but cover agile development practices comprehensively. The patterns in the toolbox have to be generic enough for enabling reuse but include sufficient details and references to further information for being practical. Thus some editorial work is needed to create and maintain a set of patterns that form a coherent toolbox.

Patterns can be categorised in various dimensions. Different classification schemes are needed because people are looking the patterns from different viewpoints. The following categorisations will be used:

Type - Leavitt's Diamond

- Organization
- Technology
- Process
- People

Organizational Scope

- Organization
- Community
- Project
- Individual
- N/A

CMMI Process Area

- Project Management
- Design
- Implementation
- Testing
- Metrics
- Quality Assurance

Content

- Working principle
- Method
- Standard

Originating Method

- XP
- Scrum
- Etc.

Patterns will be listed in pattern catalogue that serves as an index for the patterns. The pattern catalogue provides the names and the different classifications of the patterns. Pattern catalogue will be available in the appendix x.

Patterns will be documented using the template provided in the appendix X and the pattern descriptions are collected in the appendix x. The pattern descriptions in the toolbox shall contain the following parts:

Pattern Name: Short descriptive name.

Description: One paragraph description of the pattern.

Problem to Solve: Description of the dysfunctionalities that the pattern is supposed to solve.

Goal (Aka expected results): Desired outcome after successful implementation of the pattern. This may refer to some experience reports and studies.

Relationships: This section lists identified dependencies to other patterns. The dependencies may be of different nature.

The direction of the relationship may be

- Predecessor
- Successor
- Mutual

Characteristic can be:

- Catalyst
- Inhibitor

Strength can be:

- Must
- Strong
- Weak

Note: strength may vary depending on the situation.

Risks: Potential risks that should be considered when the pattern is implemented.

Helps and Hinders (aka Forces): Characteristics of the organization, processes, domain etc. that make the implementation of the pattern easier or more difficult.

Rationale: Description of the underlying mechanism that explains why/how the pattern works. Typically this can be a reference to a scientific theory and some related studies.

References: References for best sources of further information. There can be a separate list of references to practical and scientific information.

FAQ: Lessons learned from the use of the pattern. Content of this chapter varies depending on the pattern. This section provides information how to apply the pattern e.g. in different domains. It may also give some implementation guidelines.

Pattern Editor: Name and contact information of a person maintaining the pattern.

6.6 Experiences

Experiences of the use of the example methods and practices are collected using an experience collection template. The template can be found from appendix X.

7 Framework relationship to standards

Because the framework as a whole is not to be used as a method for software development, it does not make sense to map the whole framework into standards. Instead, mapping of individual example methods to standards is possible. Thus it is up to the example method developers to identify the relevant standards and to provide the mappings as a part of the example method.

An informative mapping of practices in the toolbox to CMMI process areas is provided in appendix X.

8 Framework Management

The following roles participate to the management of framework and its content.

Framework Owner is responsible for maintaining the overall structure of the framework and analyzing the change requests to it.

Pattern Owner is responsible for developing a single pattern in the framework and collecting experiences regarding to it. Each pattern has typically 2-4 pages of information. Each pattern owner typically has 2-3 patterns to edit.

Toolbox Owner is responsible for editing the patterns so that they are consistent both in format and content. Toolbox editor is also responsible for creating new patterns as needed and to help solve conflicts with overlapping patterns. Toolbox owner checks the consistency of the pattern definitions.

Method Owner is responsible for developing and maintaining one example method.

Content Owner is responsible for developing and maintaining some other defined part in the framework document or one of its appendixes.

Processes to collect and consolidate information...

9 Technical Implementation

This technical report with the appendixes will constitute the framework. The document is collated semi-annually for review and distribution purposes.

A development version is available to consortium members on a web site. The website provides the latest development versions of each pattern in the toolbox and every example method. The content owner of each pattern / methodology has access to edit the content any time during the development cycle. The patterns and methods are thus versioned each at their own pace during the development time. In addition to the edited text there is available a free-format discussion board (like wiki-wiki) connected to every pattern and method where any consortium member can comment and contribute the development.

Appendix A: Agile Practice Description Template

1 Name

Name is important part of pattern because it acts as a handle to the rest of the contents.

1. Choose a short name for the pattern
 - Think of a name that encodes the central idea of the pattern

2 Context

This chapter describes context in which it makes sense to apply solution if problem exists. Applying pattern in a wrong context might lead to failure.

1. Fill in the table below
 - Think of patterns that must be present for this pattern to make sense
2. Write down all other conditions that should match
 - Think of practices not yet described by patterns, think the types of people needed, think of project characteristics like project size

Name of related pattern	Description of relation	Rationale

3 Problem

Description of a problem that repeatedly comes up in the defined context.

1. Tell what is the problem to which solution below answers
2. Think of root causes of the problem. Tell why the problem exists
3. Tell why problem is difficult

Problem usually exists because there are conflicting forces in the context. Forces can for example be interests of some role, generic truths of sw engineering, some specific characteristic of human nature etc.

1. If possible try to describe the forces that cause the problem.

4 Solution

Solution gives instructions how to generate solution.

1. Write solution in general form so that it can be reused. Omit details that can be varied in the solution or tell the points in which the solution can be varied.
2. Think why solution works to solve the problem. Write down rationale.

5 Implementation Guide

5.1 Examples

This section contains descriptions of both successful and unsuccessful implementations. Both positive and negative examples are needed.

1. If your examples are found in experience reports or example method then fill in the table below.
2. You can also write your example in free text

Link	Description	Author contact information

5.2 Guidelines

This section contains detailed instructions how to apply in particular project or in certain setting. This section can include details omitted in solution section.

5.3 Risks

This section contains possible failure scenarios.

5.4 FAQ

Q: Write FAQ...

A: ...in this format.

6 Metadata

6.1 Pattern editor

Name and contact information of the person that is maintaining this pattern.

Name: xx

Organization: xx

E-mail: xx

6.2 Pattern origin and relation to other practices

Pattern might originate from literature. Pattern might also come up when one pattern is split up to two more general patterns. Also whole new patterns might emerge in projects.

1. Where did you get the idea of this pattern. If applicable use the references section to provide reference to literature.

6.3 Pattern classification

Classification is used in the future to search for patterns.

Patterns represent knowledge that has been cumulated throughout the use of pattern. Pattern that has successfully been used multiple times has higher confidence level than a one that has only just been discovered.

DEFINITION FOR TYPE?, SCOPE? And other classifiers!

1. Bold the right option from each classification category

Confidence level: [Just discovered / anecdotal evidence / well known and lots of research data]

Type: [Organizational / Technology / Process / People]

Organizational scope: [Organization / Community / Project / Individual / NA]

CMMI Process area: []

Content: [Working principle / Method / Standard / Prevailing condition]

6.4 References

Contains references to literature. Prefer web links in order to make tool box easy to use.

1. Write your...
2. literature etc. references in this list here


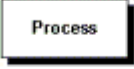





6.5 Change log

Date	Change description	Author
5.4.2005	Remember to...	
1.4.2005	..fill in this table in descending date order.	

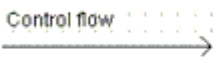



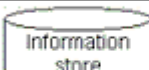
Appendix B: Recommended Practice for Process Modelling

Roles can be defined using the ARCI method - A.R.C.I. charting is a means to identify the activities, decisions, and people involved in a process. It answers the question "Who will do what has to be done?" A.R.C.I. stands for: **A**uthority, **R**esponsible, **C**ontributor & **I**nformed

Basic set

Symbol	Description
	Activity
	Process (nested)
	Work product flow (information flow)
	Role
	Start point
	End point
	Decision (branch/merge)

Additional set

Symbol	Description
	Control flow
	Milestone
	Text
	Note
	Information store

Appendix C: Principles of Agile SW Development in Embedded Systems

Version 0.01.

Collaborative knowledge creation

The organization works such a way that the opportunities for collaborative knowledge creation, where two or more people are together solving problems, are maximized. In general this means that people are using more time working together than writing alone code or documents.

References:

- Knowledge creation theories, e.g. Nonaka & Takeuchi, Knowledge Creating Company and Tuomi [1999], Corporate Knowledge
- Research on organizational communication and collaborative working techniques.

Problems to solve:

- People use time to write documents instead of doing productive activities.
- There is no shared understanding of the requirements and the system under construction.
- SW development team is making decision based on their own experiences and expectations of the customer needs. The decisions are many times wrong, because the team does not fully understand the problems that the customer has.

Rationale: Knowledge creation is a social activity that is based on communication. As a result of knowledge creation process there is a shared meaning model. Different forms of communication differ in their capability to overcome discrepancies in the pre-existing meaning models of the persons communicating. As SW engineering is knowledge creation rather than information processing according to predefined rules, communication forms that are collaborative should be preferred over document driven low interaction approaches.

Implementation guidelines:

- This principle can be applied within the team, between the team and the customer(s) or among an organizational community of practice across team (e.g. all SW architects). By involving the customer, the organization is able to collect the right requirements, assign them correct priorities, and validate them. Customer also has a better visibility to the work and achievements so far.
- There are differences also in how different communication forms can preserve information over time and how a large amount of details can be transferred exactly. Thus appropriate level of documentation is needed.

Result: Organization has a shared understanding about the problems to solve and system under construction. Everybody has an understanding of the overall technical system, their own role in it and the results of the decisions that they make.

Continuous learning

The organization looks continuously opportunities to develop the way it works. The organizational learning can be realized e.g. as processes, institutionalized practices, tools or organizational structures. In a long term this may mean building up completely new competencies. Also people-in-organization are continuously actively developing the skills they need. Organization supports them to build up the skills that they need in the future.

References:

- Learning organization research, e.g. Senge [1990], The Fifth Discipline, Sydänmaanlakka [2002], Intelligent Organization
- Double loop learning, Bateson [1973], Argyris & Schön [1978] and its application in SW engineering e.g. Basili []

Problems to solve: In changing environment also needed competencies, processes, skills and technologies are changing. If they are not developed, the organization is tomorrow facing the new problems with outdated weapons.

[Rationale: Organizations anticipate based on their accumulated knowledge what they will need in future. The anticipation can be expressed in e.g. in plans or spoken agreements and it leads to action. As a result of action there is outcome that is in accordance or conflicting with the anticipation. Possible discrepancy is information that may lead to new knowledge, in this case about what is really needed and what activities are unnecessary or performed ineffectively. This knowledge should lead to action in organizational level so that processes are changed to be more effective. CMM level 5 comprises basically the same idea, but based on measured facts rather than subjective knowledge.] Organizational learning means that improvement opportunities are identified

Implementation guidelines: Organizational learning has two important aspects: how to learn new useful things and how to stop doing harmful things. In other words learning is also about how to stop wasting effort.

Dependencies:

- Organization and its member learn through collaborative knowledge creation.
- Continuous learning is one form of incremental knowledge creation.
- Alignment of activities should be applied to continuous learning so that it contributes for achieving organization's goals.

Result: Organization maintains its competitive advantage and is even able to improve effectiveness and efficiency in a changing situation.

Alignment of activities

Activities in the organization are aligned so that they contribute for achievement of the common goals. This is achieved through e.g.:

- Creation and communication of vision and strategy
- Setting and maintaining goals that are aligned from the corporate level to individual persons.
- Establishing and managing commitments to goal, processes etc.
- The alignment of activities is demonstrated in everyday work e.g. the following ways:
- Decisions are done based on their business impact.
- Sub-optimal decisions are avoided.

References:

- Activity Theory, Leont'ev [1978], Activity, Consciousness, and Personality.
- Commitment research

Problems to solve:

- When the common vision and goals are missing teams may start to develop to different directions. Even if there has been alignment in the beginning, it may be lost when changes happen.
- Time is wasted in making the decisions if there is no clear direction (e.g. goals, policies) for the decisions.
- If the business goals are not guiding action all the time, the organization can end up using its resources not optimal way and thus wasting effort.
- If there are no commitment to the goals, people-in-organization are not putting their best effort, e.g. they are not taking deadlines seriously. When the organizational goals change, several kinds of commitment problems may occur:
 - People lost their commitment due to changes but the change is not noticed or ignored. If the commitment is lost, people are not motivated to achieve the goals.
 - Re-planning is done due to changes but no commitment is obtained for the changes.
 - Commitments are re-negotiated due to changes but plans are not updated. Thus they are giving a wrong impression.
 - Some commitments have become irrelevant due to changes, but they are not released. Thus effort is wasted to achieve irrelevant goals.

Rationale: The knowledge creation process should lead to effective action and the effectiveness can be assessed only against the goals of the particular organization. We can apply activity theory to describe how organizational motives and goals are coupled with lower level entity (e.g. team, individual) motives and goals. As there is not necessarily one-to-one relationship between the motives and goals in different levels, their structure may become very difficult to manage and the decisions in lower levels are done based on goals that are conflicting with organizational goals. If the organizational goals are articulated clearly and communicated to all organizational levels, it is possible to take them into account directly when making decisions. This supposes, however, that the individuals-in-organization have internalized the articulated goals and committed to them.

Dependencies:

- Collective knowledge creation is important mechanism of sense making of the goals set by management.
- Vision, strategies and goals in various organizational levels develop through incremental emergence of knowledge.

Result: Management, teams and individuals are committed to the common goals of the organization.

Iterative Emergence of Knowledge

Final solutions develop over a period of time. It is acknowledged that it is not possible to design everything upfront. The processes and architecture are designed so that it is possible to proceed effectively even in situations where only limited amount of information is available. The process is designed so that there is frequent and timely feedback for one's work. In other words, it is possible to detect and correct misinterpretations, ambiguities and bugs as early as possible. Processes and architecture is designed so that decision with uncertainties can be deferred to as late as possible. When making the decision later, there is probably more up-to-date information available.

References:

- Learning models by e.g. Bateson [1973], Argyris & Schön [1978], Kolb [1984], Dewey [], Engeström [1999]
- Knowledge creation theories, e.g. Nonaka & Takeuchi, Knowledge Creating Company and Tuomi [1999], Corporate Knowledge
- Evolutionary SW engineering models, e.g. Evolutionary Project Management by Gilb [], Spiral model by Bohem []

Problems to solve:

- Much effort is used to design everything upfront. As situations change, the effort is wasted because the results are not any more applicable. Work can't proceed before a certain step of the process is completed. (E.g. all stakeholders have approved the design.)
- It is noticed after a long period of time that a person has been doing wrong things or doing the things wrong way. In addition to waste his own effort, he is also causing disturbance to other persons as well. E.g. other people are building their work on wrong assumptions and thus producing flawed results or multiple persons are trying to solve problems caused by the same bug. It is also possible that new functionality or changes are built so that they work correctly when the error is present but as soon as the error is removed, they fail. Generally, more functionality built on top of erroneous assumptions, more difficult and expensive it is to fix.

Rationale: Knowledge creation is by nature an iterative and irreducible social process. New knowledge is created and acted upon continuously. Because knowledge creation process is not about finding out an objective truth but creating a unique meaning structure, it is not possible to anticipate the outcome of the process. As the knowledge is created incrementally, more accurate decisions can be done in later stages of a project. If design decisions are nailed down unnecessary, changes for better ones may be impossible or expensive.

Knowledge creation process is an iterative process where new meaning structures are built on existing ones. Because all new information is reflected on existing structures, the problems there are affecting on how the new information is perceived. In 5-A model individuals-in-organization continuously appropriate the knowledge of other individuals-in-organization and also the collective knowledge. The appropriation happens continuously e.g. when working together but it can be also institutionalized e.g. in testing or review activities. By frequent and timely appropriation it possible to correct emerging deviations in the knowledge before they have caused damage through ineffective action.

Implementation guidelines: This principle can be applied e.g. to technical architecture, organizational structure and the used processes.

Dependencies: Collective knowledge creation is one form of incremental knowledge creation. However, not all information can be created collectively but there is always some parallel work going on. The parallel work should be made available to others for appropriation as soon as possible.

Result: Organization has an appropriate organizational structure, processes and architecture. They are continuously refined to meet the changes in the environment. Possible problems or deviations are noticed and acted on quickly.

Actions with Courage

Hard decisions are taken rather soon than late. This can be applied to architecture related changes, organizational structures and used processes.

References: XP, Beck

Problems to solve: Major changes are not implemented as soon as they have become apparent. Postponing the changes makes them even harder to do later.

Rationale: Knowledge should lead to effective action. If there is no adequate courage to act upon existing knowledge, effective action is hindered.

Implementation guidelines:

Dependencies: Alignment of activities help to make fast decisions.

Results:

Pursue Simplicity

Processes, architecture and organizational structure are designed so that no unnecessary complexity is introduced. These are also aligned so that no unnecessary additional complexity is introduced.

References:

- Conway [1968] article in Datamation
- Design patterns and refactoring, e.g. Coplien, Fowler

Problems to solve: Additional complexity creates additional management and communication overhead.

Rationale: In a volatile environment simple structures are easier to maintain and modify than more complex one.

Implementation guidelines:

- There is continuous strive towards simplicity. Typical scenario in SW development is as follows: New exceptions are introduced and the complexity increases. After a few similar changes, regularity is observed in the exceptions and by applying a new design pattern it is possible to decrease the overall complexity.
- High cohesion, low coupling design is considered to be the criteria of simplicity in SW.

Dependencies: Iterative emergence of knowledge is a way to achieve simplicity.

Result: Easier to maintain SW, easier to manage organization.

Be Integer

Everyone does their best effort to give to their peers and manages as correct and complete information as possible. (E.g. schedules, possible problems)

References:

- XP
- Lean SW Development, Poppendieck, 2002

Problems to solve: It is difficult to manage the project if there is a difference between reported situation and reality.

Rationale: Knowledge creation process is collective meaning processing. All misleading information disturbs the process, because it is used as a basis for constructing new knowledge. If misleading information is given on purpose, the organization may become more skeptical for all information.

Implementation guidelines: It is important to create trust within the team, between the team and the customer, between teams and between developers and management.

Result: People-in-organization trust each other and consequently the information they receive.

Maintain Lean Inventories

Amount of undone work is kept low in all stages of process. This can be achieved by removing bottlenecks from the process or by setting clear priorities for things that shall be done and that shall be postponed.

References:

- Lean SW Development, Poppendieck, 2002
- Goldrat, Theory of Constraint
- Lean manufacturing processes

Problems to solve: SW development is slow to react change requests because there is a long to-do list.

Rationale: This principle originates from Theory of Constraint (TOC) by Goldrat and it has been applied to SW engineering although originally intended for manufacturing industry. There has been critics against the use of TOC in SW engineering because SW items (e.g. requirements, errors) are interchangeable, they vary in size & effort and they may be handled in different sequence than they arrive (e.g. prioritization). However, this principle can be interpreted meaningfully in the 5-A model. Because knowledge does not exist before it is processed and incorporated in meaning structure, the issues waiting in the queue are not yet a part of the knowledge and thus they can't affect on action. Long lead-time means that action is not based on all available information. By allocating resources so that the time to solve issues is shorter, the action will be based on more up-to-date understanding of the situation.

Dependencies:

- Alignment of activities is a prerequisite for lean inventories. It ensures that there is no sub-optimization in the inventories.
- Lean inventories principle is complementing simplicity. The removal of possible bottlenecks can be seen as one criteria of simplicity.

Result: New issues and requests can be analyzed and acted with a short lead-time.

Establish System View

The big picture of the technical environment is kept in mind when making technical and business decisions. Everybody has an understanding of the overall technical system, their own role in it and the results of the decisions that they make.

References:

- Domain engineering
- Architecture management

Problems to solve: Changes in one part of the systems make undesired side effects on other parts or at the system level.

Rationale:

Implementation guidelines:

Dependencies:

- Alignment of activities helps to maintain system view.
- Collaborative knowledge creation across team boundaries is a way to achieve a commonly shared system view.
- Simplicity helps to achieve system view.

Results: Others work is not jeopardized by making sub-optimal decisions.

Appendix D: Pattern catalogue

Version 0.01

Practice	Origin	Description	Responsible Person
Roles	XP, Scrum, etc	Team members have different roles based on their areas of responsibilities.	
Planning game	XP	The way to customer and the team to plan and communicate which tasks are to be implemented in each iterations.	
Small / short releases	XP	The product is done in iterative style and new versions are "published" rapidly.	
Metaphor	XP	Simple story of the purpose of the application.	
Simple design	XP	Tasks are solved with the simplest possible way to avoid unnecessary complexity.	
Test-driven development*	XP	Unit tests are written before the actual code.	
Acceptance testing*	XP	Customer writes the acceptance tests.	
Refactoring	XP	Removing duplication and adding simplicity.	
Pair programming	XP	Coding is done in pairs using one computer.	
Collective code ownership	XP	No one owns the code and everybody are allowed to change any parts of the code.	
Continuous integration	XP	New code is integrated as soon as it is ready.	
40-hour-week	XP	Avoiding working overtime.	
On-site customer	XP	Customer is present and available for the team to make questions etc.	

Coding standards	XP	Coding rule that everybody follows.	
Open workspace	XP	Team works in the same space without physical obstacles.	
Just rules	XP	Team has its own rules and they are followed. Rules are decided together.	
Product backlog	Scrum	Includes the tasks that needs to be done for the final product.	
Effort estimation	Scrum	The efforts are estimated and updated in an iterative way.	
Sprint	Scrum	Production is done in short sprints (~iterations)	
Sprint planning meeting	Scrum	Meeting to plan the sprint.	
Sprint backlog	Scrum	List of product backlog items that are intended to be done in the sprint.	
Daily scrum meeting	Scrum	Short daily meetings to present the progress and present problems.	
Sprint review meeting	Scrum	Review in the end of the sprint to present the results and decide following actions.	
Incremental delivery	Crystal family	Product is delivered on regular basis.	
Progress tracking	Crystal family	Progress tracking by milestones based on the software deliveries rather than written documents.	
Direct user involvement	Crystal family	Direct user involvement.	
Automated regression testing	Crystal family	Regression tests are automated.	
Product and methodology workshops	Crystal family	For tuning the product and the methodology at the beginning and the middle of each increment.	
Staging	Crystal family	Planning the tasks that are implemented in each increment.	

Revision and review	Crystal family	Iteration consists of activities: construction, demonstration and reviewing the objectives of the increment.	
Monitoring	Crystal family	Progress is measured by milestones and stability stages of the teams deliverables.	
Parallelism and flux	Crystal family	After finishing a task the next task can begin.	
Holistic diversity strategy	Crystal family	Splitting large functional teams into cross-functional groups to offer the knowledge for single teams.	
Methodology-tuning technique	Crystal family	Project interviews and workshops are used to tune the development process in the projects.	
User viewings	Crystal family	At least once per iteration the product viewing is arranged for the user(s).	
Reflection workshop	Crystal family	Teams should hold pre- and post-increment workshops.	
Domain object modeling	FDD	Exploration the domain of the problem, adding the results in the framework where the features are added).	
Developing by feature	FDD	Development is done through a list of small customer-valued features.	
Individual class (code) ownership	FDD	Every piece of code has an owner that modifies it.	
Feature teams	FDD	Small teams are used for the development.	
Inspection	FDD	Use of best-known defect-detection mechanisms.	
Regular builds	FDD	Building of the system is done regularly to act as a baseline for new systems to be added	
Configuration management	FDD	Configuration management is used to track the historical data.	

Progress reporting	FDD	Progress is reported based on the completed work.	
Develop software iteratively	RUP	development is done in small increments and short iterations.	
Manage requirements	RUP	Requirements are identified and updated.	
Use component-based architectures	RUP	Software is built using isolated components.	
Visually model software	RUP	Models of the software are done to support the understanding.	
Verify software quality	RUP	Testing is done during each iteration.	
Control changes to software	RUP	Changes to the requirements are managed and their effect to the software needs to be traceable.	
Iterative development	ASD	The development is done in development cycles.	
Component-based planning	ASD	The planning is based on components rather than tasks that needs to be done	
Cycle reviews *	ASD	Review of the whole project, team and the processes in the end of each cycle.	
Customer focus group reviews	ASD	Customer group that explore the application and makes change requests.	
Agile modeling	AM	Method for modeling: Use simple tools, model when needed and only what is needed.	
Display models publicly*	AM	Display the models in public.	
RaPiD7*		Documents are done in the workshops where multiple stakeholders are present.	

SEED*		Workshops are used to manage requirements and architecture.	
Integration camp*		Software is integrated in the workshops.	
Planning day, Working day, Release day *	MobileD	Iterations includes different activities in the beginning and in the end of the iteration.	
Information radiator *		The tasks are collected to a notes that are moved on a poster implying the state of the task.	

Appendix E: Mapping of patterns to CMMI process areas

Level 2. Software project planning, monitoring and control

Planning Game
Short releases, incremental development
40-hour workweek
Open workspace
On-Site Customer

Level 2. Project Monitoring and Control

Iteration review
Information radiators
Daily meetings/ Daily wrap-ups/ Scrum stand-up meetings
Planning Game, Risk monitoring
Use of Agile Metrics
Close communication with customer (on-site customer)
Project Milestones in Agile development

Level 2. Quality Assurance

Post-Iteration Workshops
Agile assessment
Refactoring
Coding standards
Quality audits before the releasing

Level 2. Configuration Management

Incremental release planning (for whole product)/ Small releases/ sprints
Product and Sprint backlogs
Change request tracking

Level 3. Engineering Process Areas

Methaphora
Teamwork for specifications and documentation
Agile Trials in 0-iteration
Architecture line
Simple design
Pair Programming
Collective code ownership

Level 3. Verification (Testing and integration)

Unit testing
Test first development
Customer participation in testing
Continuous Integration