

# Framework of Agile Patterns

Teodora Bozheva and Maria Elisa Gallo

European Software Institute, Parque Tecnológico Edif. 204,  
48170 Zamudio (Bizkaia), Spain  
{Teodora.Bozheva, MariaElisa.Gallo}@esi.es  
<http://www.esi.es>

**Abstract.** The variety of agile methods and their similarity could be a problem for software engineers to select a single or a number of methods and to properly execute them in a project. A pattern describes a problem, which typically occurs under certain circumstances and a basic approach to solve it providing opportunities to adapt the solution to the problem. The agile patterns, described herein, are based on the principles and practices of the best known agile methodologies. While individual practices included in any of these methods vary, they all have particular objectives and related to them activities. Therefore, every pattern is described as to show the core solution to a particular problem. Special attention is paid to the rationale for applying the agile patterns: what are the business drivers to adopting them; in what cases do they bring benefits; how could they be introduced in an organization.

## 1 Introduction

Nowadays lots of organizations face the need to adapt quickly to modifications requested by their customers, changes on the market or challenges from competitors. This happens in small as well as in large organizations, in ones following standard (ISO 9001:2000, CMMI) or their own processes. These business needs force the companies evaluate how the agile methods could address their necessities.

Agile methods recognize that any project, team and organization has its unique peculiarities and respond to the specific needs via business value based prioritization, short feedback cycles and quality-focused development. When appropriately applied the agile practices bring a number of business benefits as better project adaptability and reaction to changes, reduced production costs, improved product quality and increased user satisfaction with the final solution.

The agile methods differ in the approaches to software development and management they propose. Some focus more heavily on project management and collaboration practices. These include Adaptive Software Development (ASD) [6], Scrum [7], Lean Development (LD) [10] and DSDM [8]. Others, such as eXtreme Programming (XP) [3], Feature-driven Development (FDD) [9] and Agile Modeling (AM) [5], focus more extensively on software implementation practices. Nevertheless, all the methods stick to the principles of maintaining good

understanding of the project objectives, scope and constraints, developing software in short, feature-drive iterations, receiving constant feedback from the customer and the developers, and focusing on the delivery of business value.

An important issue in defining an organizational process is that all the elements it consists of reflect properly the specifics of the environment, in which the process will be implemented. When selecting an agile method, the business and organizational context, in which it will be applied, determines the benefits that could be achieved for a project and for an organization. In their book [1] B. Boehm and R. Turner have defined the “home grounds” in which agile and disciplined methods are most successful. Additionally, they define five factors, which help the organization determine whether they are in either the agile or disciplined area, or somewhere in between. These are size, criticality, personnel, dynamism, and culture.

Our work on applying agile practices in different organizational contexts inspired the development of the framework of agile patterns, presented in this paper. The idea is instead of providing a complete method, which might not be fully applicable in any situation, to provide a set of patterns addressing different aspects of the software development process, which could be combined in such a way as to fit to the peculiarity of a project. The patterns are derived from the most widely known lightweight methods XP, Scrum, FDD, AM, LD, and ASD.

Implementing a software development process based on patterns has several advantages:

- The patterns address activities performed by software engineers and project managers who are accustomed to using well structured information like pattern definitions.
- Patterns describe individual practices in a general enough way to be applied in different situations. Therefore they can be easily tried out and included in definitions of new processes. Adopting a small set of new practices gives a more profound understanding of the practices themselves and of the benefits from applying them together, which facilitates the continuous process improvement.
- As each pattern is selected and adapted as to best fit a project and organizational context, the whole process will be more suitable for that context than any general one.

We describe the framework of agile patterns in section 2 and in section 3 we present lessons learnt from applying the patterns in the industry.

## 2 Framework of Agile Patterns

A pattern describes a problem, which typically occurs under certain circumstances. It also describes a basic approach to solve the problem providing opportunities to adapt the solution to the particular problem context. In general, a pattern has three essential elements: problem, solution and consequences. Each solution consists of activities that, when collectively applied, resolve the problem. The solution is abstract enough to make it possible to apply it in different situations. The consequences are results and trade-offs of applying the pattern.

Three key terms take part in the agile methods: *practices*, *concepts* and *principles*. *Practices* describe specific actions that are performed in the whole process of software development, e.g. create product backlog (SCRUM). *Concepts* describe the attributes of an item, e.g. a project plan. *Principles* are fundamental guidelines concerning software development activities, e.g. empower the team (LD).

To be coherent with the agile methodologies the framework of agile patterns (FAP) includes definitions of three types of patterns: practice patterns, concepts and principles.

## 2.1 Practice Patterns

In the FAP each agile pattern is described by means of the following attributes:

- **Intent:** a short description of what the objective is;
- **Origin:** methodologies, from which the pattern originates;
- **Category** to which the pattern belongs. With respect to the type of issues addressed, the patterns are grouped in the following categories: *Project and Requirements Management, Design, Implementation and Testing, Resource Management, Contract Management and Software Process Improvement*.
- **Application scenario:** context, in which the pattern is to be applied;
- **Roles:** people involved in carrying out the pattern and their responsibilities;
- Main and alternative **Activities** that constitute the pattern. Activities can invoke other patterns;
- **Tools** that support the pattern execution;
- **Guidelines** for performing the activities including suggestions for making a decision about which alternative solution to choose when.

This structure is closest to the one proposed by E. Gamma in [2]. Compared to the classic pattern definition (problem-solution-consequences), Intent and Application scenario correspond to the problem attribute. Activities matches to solution. Some patterns provide alternative solutions to the same problem. This typically happens when the problem is addressed by more than one agile method and different solutions to it are proposed. Guidelines include hints for performing the activities and the consequences from them. An example of a pattern is:

### ***CodeIntegrator***

**Intent:** To have working code at the end of every day. In a software development environment with collective code ownership, the idea is to build the system every day, after a very small batch of work has been done by each of the developers.

**Origin:** *LD:* Synch and Stabilise (Daily build and smoke test); *XP:* Integrate often

**Category:** Implementation & Testing

**Application scenario:** After implementing a piece of code

**Roles:** Developers

### ***Activities***

- Check out source code from the configuration management system.
- Put together the newly implemented and the existing code.
- Check to see, if anyone else has made changes to the same code, and if so, resolve the conflicts by applying CodeImplementer.

4 Teodora Bozheva and Maria Elisa Gallo

- Apply AcceptanceTester.
- Check in the new code.



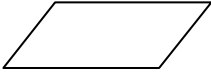

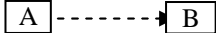
**Tools:** Version control tools support this activity.

**Guidelines:**

- Continuous integration avoids or detects compatibility problems early. If changes are integrated in small batches, it will be infinitely easier to detect and fix problems.
- A single integration point (computer) has to be defined.
- Every developer is responsible for integrating his/her own code always when there is a reasonable break. This could be when all the unit tests run at 100% or some smaller portion of the planned functionality is finished. Only one developer integrates at a given moment and after only a few hours of coding.
- All the tests have to pass successfully after integrating the system. Each integration results in a running system. Integration happens every 1-5 hours, at least once a day.

Apart from the natural language description of the patterns, every category is graphically illustrated showing which patterns, concepts and principles it includes, and the relationships between them.

On the graphics the following symbols are used:

Symbol	Meaning
	Principle
	Pattern
	Concept
	"invokes"
	Pattern A supports B, but it is optional to use A when implementing B

As an example, let's consider the *Implementation and Testing* category (Fig.1).

The patterns and their intents, which belong to this category, are the following ones.

*Code Implementer:* Implement code

*FDDCoder:* Implement defect-free code following FDD

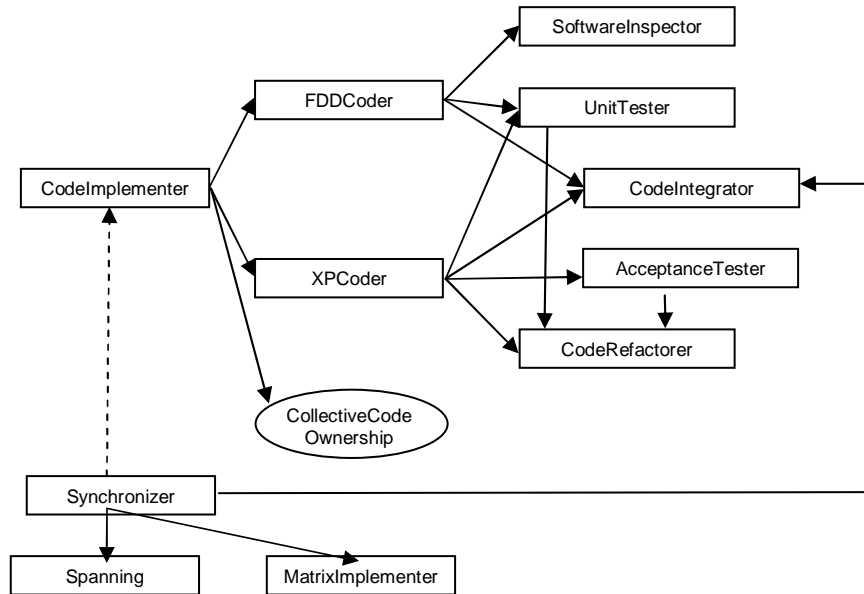
*XPCode:* Implement defect-free code following XP

*SoftwareInspector;* Find out defects in project work products

*UnitTester:* Define and execute in an automated way unit tests for a module of code.

*AcceptanceTester:* Define and execute in an automated way tests written by the customer to show that her requirements have been implemented correctly

*CodeIntegrator*: To have working code at the end of every day. In a software development environment with collective code ownership, the idea is to build the system every day, after a very small batch of work has been done by each of the developers



**Figure 1: Implementation & Testing category**

*CodeRefactorer*: Maintain good quality of the code

*Synchronizer*: Maintain the code developed and owned by several people synchronised

*Spanning*: Synchronize the work of several teams using a simple spanning application that allows getting a real understanding of the strengths and weakness of alternative solutions.

*MatrixImplementer*: Synchronize the work of several teams by letting them sketch out an overall architecture, and then develop separate components or subsystems.

The *CollectiveCodeOwnership* principle belongs to this category, and there are no concepts in it.

A brief description of the other pattern categories is provided in the Annex.

## 2.2 Concepts

Concept is a definition of a class of items that consists of characteristics or essential features of the class. For consistency reason the concepts in the FAP are described by a subset of the attributes of a practice pattern. The principal distinction of a concept from a practice is that there are no activities associated with the concept.

For instance, *ProjectPlan* is a concept describing the attributes of a good project plan in accordance with the philosophy of the agile methodologies.

***ProjectPlan***

***Intent:*** Serves as a focal point and quick reminder of the most important elements about the project.

***Origin:*** ASD: Project Data Sheet; XP: Release plan; FDD: Development plan

***Application scenario:*** Project planning

***Roles:*** Customer: makes business decision (scope, priorities, release planning)

Developers: make technical decisions (effort estimations, risks)

Project Leader: makes the planning

***Definition:*** The Project Plan is one-page summary of the key information about the project. The Project Plan includes the following details:

- Project objectives statement
- Overall Architecture
- Major project milestones
- Core team members

The project objectives statement should be specific and short (25 words or less), and it should include important scope, schedule, and resource information.

***Guidelines:*** In FDD the development plan consists of:

- Feature sets with completion dates
- Major feature sets with completion dates derived from the last completion date of their respective feature sets
- Chief Programmers assigned to feature sets
- The list of classes and the developers that own them

Eleven concepts are defined and used in the FAP.

*Mission* describes what constitutes a software development project. It consists of *Vision*, *ProductSpecificationOutline* and *ProjectPlan*.

*Vision* provides a short definition of key business objectives, product specifications, and market positioning.

*ProductSpecificationOutline* describes the features of a product in enough detail so that developers can understand the project scope, create a more detailed adaptive iteration plan, and estimate the general magnitude of the development effort.

*ProjectPlan* serves as a focal point and quick reminder of the most important elements of a project.

*IterationPlan* describes the requirements to be implemented within an iteration.

*DesignDocument* is a sufficient enough description of a software product design, which facilitates the verification of the output of the design activities.

*DailyMeeting* is a short meeting of a project team, which purpose is to communicate the current status of the project and problems encountered.

*FixedPart* indicates the content of a fixed part of an agile contract.

*VariablePart* indicates the content of a variable part of an agile contract.

*Team* is the software development team.

*Waste* in software development is everything that does not deliver business value to the customer (development of not explicitly required features, partially done work, paperwork not directly needed for the development processes, handoffs, defects, switching between different projects, waiting for a project/task to start).

### 2.3 Principles

Principle is a set of fundamental guidelines concerning the software development activities. The principles are people-oriented and flexible, offering generative rules. Again for consistency reason the principles in FAP are described by a subset of practice pattern attributes: Intent, Origin and Guidelines. An example of a principle is

***CollectiveCodeOwnership***

***Intent:*** The code is collectively owned by the developers. Anyone change it. The programmers use a coding standard to enforce a common style.

***Origin:*** XP: Collective Code Ownership

***Guidelines:*** Collective code ownership is more reliable than putting a single person in charge of watching specific pieces of code, especially because, if a person leaves the project at some time, the other project team members will know the code he has implemented and will be ready to continue his work.

We consider seven principles most important among the ones defined in the investigated methods. *Collective code ownership* is from XP and the rest of the principles originate from Lean Development.

***Avoid sub-optimization.*** Instead of optimizing the performance of small project parts, optimize the complete system, i.e. focus developers on what's important, namely meeting the customer's business needs, not on building a product with the excellent characteristics from technology point of view only.

***Decide Late.*** Take decisions as late as possible, reducing in this way the risk of making mistakes due to insufficient information.

***Deliver Fast.*** Provide rapid delivery to customers. This often translates to increased business flexibility.

***Empower the Team.*** Move decision-making to the lowest possible level in an organization, while developing the capacity of those people to make decisions wisely.

***Queueing Theory.*** Optimize resource management as to reduce the time spent for waiting for a resource to start working on a task.

***Simple Rules.*** Chose a small number of strategically significant processes and craft a few simple rules to guide them.

### 2.4 FAP and Other Related Approaches

The patterns approach is not new in the software development area. However, most of its applications address object-oriented design and implementation of software. [2] defines design patterns. Wiki (<http://www.c2.com>) provides a catalog of object-oriented patterns. There are plenty of books on this subject and the Pattern Languages of Programs conferences (<http://hillside.net>) dedicated to it too.

These are valuable sources of knowledge for the software developers. Nevertheless, it proves that being able to implement good software is not enough in the e-era, which demands additional capabilities to perform in a flexible and rapid manner. Therefore the focus of our work is on the agile practices for software development and management. At the time being the repository of the agile patterns is being developed and piloted within the ITEA AGILE project (ITEA IP030003; <http://www.agile-itea.org>).

### 3 Guidelines for Applying the Framework of Agile Patterns

From software developer's perspective the key benefits of the patterns are the experience-based guidelines and the rationale for patterns implementation, which complement the definitions derived from the literature. Concerning practical experience with the agile practices we gathered valuable information from seven projects performed in different organizations within the *eXpert* project (IST-2001-34488; <http://www.esi.es/Expert>). The trials were focused on applying XP practices in e-commerce and e-business application development. The main objectives for the projects were to evaluate how the agile practices contribute to increasing the productivity and the efficiency of the software engineers, and to improving the quality of the products they develop<sup>1</sup>. Other relevant experience has been coming from companies to which we provide consultancy services on software process improvement. Currently the patterns are being experimented in the ITEA AGILE project.

#### 3.1 Who should apply the agile patterns

Similarly to all lightweight methodologies the agile patterns are most appropriate for highly dynamic projects. In all the cases, mentioned above, the business needs forced the companies evaluate how agile practices could address their necessities. Out of 13 companies 7 wanted to increase their responsiveness to the changing customer's or market needs; 6 were looking for ways to decrease their time-to-market. Two companies had to follow the rigorous *Metrica-3* methodology, and were looking for a way to perform some activities more flexibly, remaining compliant with the basic methodology. Three companies were applying ISO or CMMI standard. The other ones had their own processes established.

Among the specific problems, the organizations had, were provision of unclear requirements and frequent changes to them; incorrect effort estimation, which later caused difficulties to deliver the product on time without working extra hours or adding more resources; inefficient project management. One of the project teams said that they were feeling like a fire brigade, because whenever their customer needed to quickly release a product, he contacted them and expected their professional solution on time and within budget. Although all this sounds trivial, the companies did not believe that the traditional approaches would help them and started investigating more lightweight ones.

The agile patterns are successfully adopted by small teams having development experience and motivation to maintain good communication with the customer and to deliver software with low defect rate in a short time. It is easier to introduce the patterns in organizations with a relatively flat hierarchy, because the direct communication to the management is important for the success of the projects.

---

<sup>1</sup> For the sake of completeness, the results from the experiments are as follows: Productivity increased up to 73%. One company decreased its productivity; Schedule deviation reduced between 7% and 38%; Cost deviation decreased up to 31%. Only one company increased its cost deviation; Defect rates reduced between 10% and 83%.

With respect to CMMI and ISO-certified organizations the agility of their processes can be increased by means of agile patterns and the typical work products could be developed in a more flexible and efficient manner.

### 3.2 When should be agile patterns applied

Our experience shows that agile patterns are successfully applicable in the following cases:

- Projects, in which the client only has a broad-brush picture of the envisioned system without knowing its detailed frames and final features. Using patterns from the category of *Project & Requirements Management* reduces the time and effort spent on initial customer requirements analysis and involves the client in the implementation of the system features.
- Projects, which development includes exploration and application of new or less known technologies. In such cases the *Iterative&IncrementalModeller*, *InTeamModeller* and *CodeImplementer* patterns, applied in short iterations, are of particular use.
- Projects, which outcome is a critical success factor for the organization's business. This implies close tracking of the results, the schedule and budget.
- The client and the development team have established high confidence relationship and mutual trust, which brings additional success factor for the implementation of a new process. Constant awareness on the project status increases the client's confidence in the final result and in the agile practices.
- Teams applying CMMI or ISO who are interested in increasing the agility of their processes. In such cases the developers should select patterns to appropriately substitute practices they currently perform and to adapt the patterns to the other process activities.

### 3.3 How to adopt the agile patterns

The adoption of agile patterns has to be done gradually, and it has to be taken into account that the better the philosophy of the agile methodologies is accepted and applied in an organization, the higher the benefit from the practices used.

These are typically the steps to start applying agile patterns:

#### 1) Identify Changes to be Made

Apply the *WasteEliminator* to identify processes or parts of processes, which agility has to be increased. Identify agile patterns that could be used to improve the current activities. Define how the selected agile patterns will be adjusted to the practices, which are already in place.

#### 2) Customize a patterns-based process to a project and the team

Building a process from patterns is like using a Lego construction set. First, one should have an idea of what he wants to build up and afterwards to start constructing it combining different pieces. In the context of software development it translates to identifying which activities should be made more lightweight and applying respective patterns to achieve them. Since the approach is very much dependent on the team

culture and on the specifics of the projects, the application of the patterns has to be analyzed and adapted to a particular context.

3) Introduce selected patterns

Explain to or train all the team members how to apply the selected patterns. A gradual transition from a heavyweight to an agile process make the changes easier for the development team. We have supported several projects in applying the agile patterns and the most important lessons learned by the development teams are as follows:

- FAP should not be adopted using the “Big Bang” approach. That is the team should not try to apply at once all the patterns covering a complete development process because some of the practices are very different from what most of the developers are used to do, e.g. write test cases before the code. Instead, the team should try to adopt the patterns sequentially, following the natural project lifecycle. However, the patterns are dependent on each other; therefore all selected ones have to be put in place at some point.
- The good communication with the customer is a key factor. The foundation of an agile approach is in the close customer involvement into the project creation from the first steps to the last test performed. It is not so important to have the customer really on-site. An alternative is Customer on-demand (by phone, mail, meetings). The important thing is that if the customer gives the team rapid feedback when required. If the customer does not provide prompt feedback on project issues then the project is in trouble. It can spoil all other efforts, no matter how hard the team tries, because there will be no way they can be sure that the product they build is the product the customer wants.
- Automate! Use tools wherever possible. They save a lot of manual effort and increase the productivity a lot. There are lots of tools that aid the unit testing, acceptance testing, refactoring, requirements management, defects tracking, effort tracking, etc. Lots of them are open source or free, so they will not compromise the project budget. The effort needed to start using them is usually very small compared to the benefits they provide.

## 4 Conclusions

Today's enterprise solutions are complex, time critical and developed against rapidly changing business needs. The framework of agile patterns helps software engineers and practitioners to optimize the software development by selecting and putting together practices that fit the peculiarities of their projects and teams better than whole methods.

This approach provides a superior synchronization throughout a project and an organization, because the positive results are quickly seen and motivate people to go in the direction of flexibility, adaptability and responsiveness.

## References

1. Boehm B., Turner R.: *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison Wesley (2003)

2. Gamma E., et al: Design Patterns, Addison-Wesley (1995)
3. Beck K: Extreme Programming Explained: Embrace Change, Addison-Wesley (2000)
4. Poppendieck M., Poppendieck T.: Lean Software Development: An Agile Toolkit for Software Development Managers, Addison-Wesley (2003)
5. Scott W. Ambler: Agile Modelling, John Wiley&Sons, Inc. (2002)
6. Highsmith J.A: Adaptive Software Development: A Collaborative Approach to Managing Complex Systems, Dorset House Publishing (2000)
7. Schwaber K., Beedle M.: Agile Software development with Scrum, Prentice Hall (2002)
8. <http://www.dsdm.org>
9. Palmer S., Felsing J., A Practical Guide to Feature-Driven Development, Prentice Hall (2002)
10. Poppendieck M., Poppendieck T.: Lean Software Development: An Agile Toolkit for Software Development Managers, Addison-Wesley (2002)

## Appendix: Patterns catalog

The patterns from the Implementation and Testing category (discussed in the paper) and the Communication category diagram are not included in this appendix for the sake of space limits.

*AgileContractCreator*. Define a contract that provides a software provider some degree of flexibility in any development parameter: schedule, cost, scope or quality.

*AgileDocumenter*. Create and maintain models and design documentation.

*Communicator*. Maintain good communication between project stakeholders

*CustomerFeedbackIncreaser*. Increase the feedback from the customers to development teams by holding a customer focus group at the end of each iteration

*Designer*. Design a software product in an iterative and incremental way.

*DesignRefactorer*. Improve the design of a software product.

*DevelopmentFeedbackIncreaser*. Increase the feedback from the development team to the management.

*FeatureDesigner*. Identify and specify classes to be involved in the design of a feature

*FeedbackIncreaser*. Increase feedback from developers and customers

*IterationReviewer*. Review an application, find, record and document customer changes requested to be implemented in the next iteration.

*Iterative & Incremental Modeller*. Perform iterative and incremental modeling.

*Iteration Planner*. Make a plan of the requirements to be implemented in one iteration

*InTeamModeller*. Enable effective teamwork and communication within the development team and with the project stakeholders.

*Mission Creator*. Create and document project mission. The related artifacts need to answer three questions: What is this project about? Why should we do this project? How should we do this project?

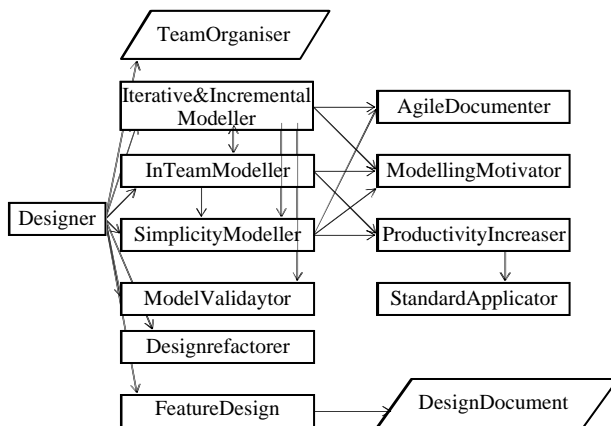
*MissionValuesSharer*. Building a shared vision and responsibility for achieving the project mission.

*ModellingMotivator*. Better understand and communicate the models. Compare design alternatives to identify the simplest solution that meets the requirements.

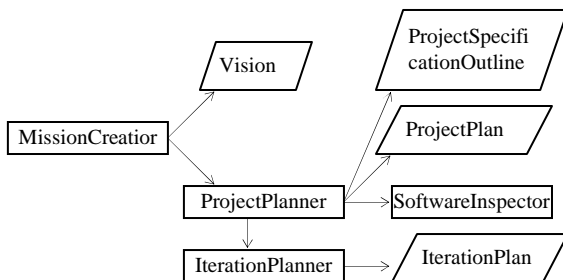
*ModelValidator*. Validate the design work by considering how it will be tested and proving it by code.

*ProductFeedbackInceaser.* Increase development team’s feedback about the product  
*ProductivityInceaser.* Optimise the work of a team as to increase its productivity.  
*ProjectCloser.* Help people learn from experience and anticipate the future.  
*Project Planner.* Make a plan of the requirements to be implemented in a project  
*SimplicityModeller.* Enable simplicity within the modeling effort. That means keeping the actual content of the models (requirements, architecture, design) as simple as possible, depicting models simply, using simple tools.  
*SoftwareInspector.* Find defects and accelerate team learning  
*StandardApplicator.* Model and code according to agreed standards. This ensures that the models and the code communicate as clearly as possible.  
*TeamFeedbackInceaser.* Increase the feedback within the team.  
*ValueMapper.* Create a flow diagram of an activity granulating it to atomic tasks and specifying the time /effort/expenses spend for the performance of each atomic task.  
*ValueTracker.* Identify the points in a process or activity, where effort is spent without generating value for the customer.  
*WasteEliminator.* Reduce the activities, which take part in a process, but do not generate value for the customer, i.e. do not contribute to the final result.

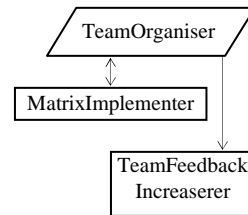
**Design**



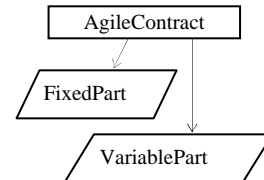
**Project and Requirements Management**



**Resource Organization**



**Contract Management**



**Software Process Improvement**

