A Framework for Harnessing the Best of Both Worlds in Software Project Management: Agile and Traditional

Adam A. Noureddine
adam@microlead.com
Microlead Business Solutions, 7240 Glenview Drive
Richland Hills, Texas 76180, USA

Meledath Damodaran
damodaranm@uhv.edu
University of Houston – Victoria, 3007 N. Ben Wilson
Victoria, Texas 77901, USA

Samira Younes
samirayounes@sbcglobal.net
Lockheed Martin Corporation, 7025 Chase Ridge Trail
Fort Worth, TX 76137, USA

Abstract

According to formal project management methodology, projects go through three key phases. Out of the three, the executing phase occupies the largest portion of the project life span in which most of the work needed to achieve the objectives of the project is actually done. In software projects, executing is the software engineering process. Several methodologies of engineering processes have been established, each having a set of advantages and disadvantages based on factors such as the size of the project, complexity level, team competence, etc. The single focus these methodologies often have – such as traditional or agile, present a major challenge for software project managers. This paper proposes a framework that enables the project manager to harness the best of known engineering processes in an agile, but disciplined, manner. The framework provides an effective balance between the need for certainty and the need for agility in software project management.

Keywords: agile software project management, traditional software project management, software engineering process.

1. INTRODUCTION

In a paper titled "Why Large IT Projects Fail?" Peter Henderson (2006) argues that "requirements drift" is a major source of problems, amongst other factors, facing Information Technology projects. His conclusion points to two issues: uncertainty and executing. Uncertainty represents the inevitability of change in many aspects of the project including requirements, circumstances, and stakeholders. It is worth noting that uncertainty does not apply to all aspects of the project. For example, requirements

such as producing high quality work, and finishing on-time and within budget, remain as requirements regardless of the uncertainty level. Executing in software project management is the software engineering process and it often consumes most project resources.

As requirements drift due to uncertainty and change, their effect is most evident during project execution. The drift often leads to newly formed requirements and causes misalignment with the engineering process selected for the project. As a result, the team is either not able to execute the engineering process properly, or the process is no longer capable of addressing project needs. This indicates that most failures can be attributed to the engineering process selected.

Several engineering processes have been established each having its own advantages and disadvantages based on a number of factors such as the size of the project, complexity level, staff competence, etc. The challenge for software project managers is to select the engineering process that fits the needs of the project and can be executed efficiently. However, the challenge is complicated further by two competing priorities: the need for certainty by following rigid and formal engineering processes, and the need to remain agile to deal with drift in requirements and uncertainty.

Most processes are designed to address specific project needs and circumstances. Furthermore, it is rare that a process can be applied fully without modification to the project (Pressman, 2005.) The project manager must adjust the selected process to better fit the needs of the project. As this seems reasonable from the point of view of increasing the effectiveness of the process, a closer examination shows it to be more like a step in the dark. The modified process has not been tested before and thus it should be considered as a new approach, not a tried-and-true one. Project managers may be oblivious to the fact they are introducing new risks in the executing phase that could increase the chance of project failure.

There's a need for a framework that enables the project manager to harness the best of known engineering processes in an agile, but disciplined, manner. Agility is important to deal with change and uncertainty, and discipline is important to establish plans, manage deviations, and meet responsibilities and constraints. The framework must address three key issues for the project manager. First, it must provide a workable balance between the need for certainty and the need for agility in software project management. Second, it must define a software engineering process that is effective regardless of the characteristics of the project. And third, it must address the needs of the key stakeholders, namely the project manager, the engineering team, the sponsors, and most importantly, the customer.

The paper provides an overview of software project management and related engineering processes, then, it explains the Disciplined Agility Framework and discusses its key benefits.

2. SOFTWARE PROJECT MANAGEMENT

Project management, by definition, is a progressive endeavor where clarity of the tasks and work products evolve over a defined period of time. Projects often start with broad specifications, and as time passes, the specifications are detailed and become clearer. The modern management of projects includes three key phases: initiating and planning, executing, and closing, where each phase produces specific work products and outcomes. Normally the executing phase requires the most resources and time, followed by the planning phase (Project Management Institute, 2004.)

Despite all the progress in project management, projects continue to fail at staggering rates. The classic study conducted by the Standish Group in 1995 showed that success rate in IT projects hovered around 16%. Project management techniques have improved dramatically over the last decade, but the failure rate still remains at an alarming level. The Standish Group just released the summary version of what they call the "2009 CHAOS Report." This report tracks project failure rates across a broad range of companies and industries, all involving software projects, with approximately 50% being entirely developed from scratch, and the remaining involving various combinations of purchased/modified/in-house developed components. From their press release: "This year's results show a marked decrease in project success rates, with 32% of all projects succeeding which are delivered on time, on budget, with required features and

functions" says Jim Johnson, chairman of The Standish Group, "44% were challenged which are late, over budget, and/or with less than the required features and functions and 24% failed which are cancelled prior to completion or delivered and never used." (The Standish Group, 1995; The Standish Group, 2009.)

Project success and failure are related to some of the key attributes of the project, which in most cases are:

- Scope: specifies the objectives of the project and what needs to be done.
- Time: specifies the amount of time allotted to the project and its starting and ending dates.
- Resources: specifies the human and physical resources allocated for the projects.
- Uncertainty: represents the unforeseen circumstances that can affect factors such as time and cost. There are many sources of uncertainty including changing requirements, circumstances, and having to deal with business and technical issues that have not been addressed before

(Project Management Institute, 2004.)

Formal software project management addresses uncertainty and change factors during planning by selecting the proper development process for the executing phase. Since most time and money is spent on executing, it is reasonable to conclude that projects fail or succeed during the executing phase of the project based on their ability to deal with uncertainty and change (Schwalbe, 2006.) The most devastating problems to software projects take place during execution (Brooks, 1975.) Tasks not done on time, not done properly, or not done at all; resources over-used; tasks taking too long; are all examples of problems that take place during the executing phase and cause the project to fail. Project execution for software projects comprises of five key phases: analysis, design, construction, testing, and deployment. Many processes have been established to manage project execution and are generally divided between two major schools of thought, or methodologies: the agile and traditional methods (Pressman, 2005.)

3. SOFTWARE ENGINEERING PROCESSES

Four leading software engineering processes can be identified to cover the spectrum of agile and traditional methods:

- Waterfall: follows a systematic and linear approach to software development. It starts with planning and then progresses through modeling, construction, and deployment in sequence (Royce, 1970.)
- Scrum: organizes small working teams and yields frequent software increments according to a prioritized list of requirements (Schwaber, 2004.)
- Extreme Programming (XP): uses an object oriented approach in its software development and focuses on producing a working product early-on (Beck, 1999).
- Unified Process (UP): draws on the best features of agile and traditional methods. It is use-case driven, architecture-centric, and incremental (Ambler, 2002).

Software engineering processes are usually part of a project with specific objectives determined by sponsors and customers. They represent the executing phase of a software project in which most of the work relevant to the objectives of the project is actually done. In other words, software engineering processes are executed in a defined context, and hardly ever in a vacuum. Any discussion of engineering processes, must therefore, take into consideration the context. Software projects have common context that can be defined regardless of the size, complexity, timeframe, and expertise. Successful software projects have the following common context (Pressman, 2005):

- Finish on time and within budget: this is a basic requirement for all types of projects.
- Customer satisfaction with features and quality: this represents the ultimate test for successful projects.
- Clarity in the design to enable future development: a software product that's difficult to modify will risk be-

ing outdated and inadequate for customers' changing needs.

Based on this common context, the following are some disadvantages of the leading engineering processes:

- Waterfall: susceptible to change and uncertainty. It may lead to project failure when taking in consideration the fact that change is inevitable (Royce, 1970.)
- Scrum: does not provide adequate design documentation necessary for future development. It may not work well with projects that require high level of innovation because its focus is on bringing order to the development process (Schwaber, 2004.)
- Extreme Programming (XP): does not provide adequate design documentation necessary for future development. Marred by chaos that does not work with complex and long running projects (Beck, 1999).
- Unified Process (UP): too much emphasis on design modeling in the early stages of the project which hinders the ability to produce a working product early on (Ambler, 2002).

The following are some advantages of the leading engineering processes:

- Waterfall: attempts to clarify requirements and produce complete design documents early on to reduce uncertainty (Royce, 1970).
- Scrum: cuts through project complexity and brings order from chaos by enabling a team to organize itself, which allows a particularly productive order to emerge (Schwaber, 2004).
- Extreme Programming: produces working software very early in the development process and allows greater freedom for the development team to innovate (Beck, 1999).
- Unified Process: provides a solid planning model for the software project in the early stage that complies with formal project management (Ambler, 2002).

4. DISCIPLINED AGILITY PROCESS FRAMEWORK

Due to the evolutionary nature of software projects and because of changing markets and evolving technology, software project feature sets amount to moving targets. The project plans start at high levels and progress towards detailed definitions; requirements are initially vaguely defined and are clarified over a period of time; at the onset, only key stakeholders are involved, but more and more participate as the project evolves. Levels of uncertainty also change over the life of the project. Decision-making on a software project progresses from coarse to fine. The project team cannot make firm decisions about a phase in the development process until it has completed the one before it (McConnell, 1998).

As uncertainty changes during the project, the development approach should also change. In many cases, what is needed is a management approach that enables the project to maintain a high level of agility to deal with uncertainty, while gradually helping to bring order and discipline to the engineering process.

Major Project Phases

In formal project management, the project goes through three major phases: planning, executing, and closing (Schwalbe, 2006). During the planning phase, the project is initiated and planning documents are produced. At this stage, the plans also include a set of requirements for the software that should be considered as preliminary due to uncertainty and the evolutionary nature of software projects. As the project enters the executing phase, the team gathers more requirements and refines the project plans accordingly. During this phase, the software is actually constructed, documented, and delivered. Finally, the project is closed with a set of tasks including acceptance and learning (Figure 1).

During the major phases, key aspects of the project evolve. Stakeholder involvement changes as more of them are identified and their input and participation is sought. The software requirement set evolves and becomes clearer and more defined as work begins and stakeholders get involved. With this normal evolution, uncertainty is usually at its highest level at the beginning of the

executing phase, and gradually diminishes as more aspects of the project are clarified. As a result, the software engineering needs and focus also evolve accordingly.

Phases in Executing the Project

To effectively deal with the progressive nature of executing a project, the executing phase can be broken down into three key sub-phases based on project needs: innovation, organization, and definition (Figure 2). This is in addition to the planning and closure phases that are standard in every project. We now describe each phase.

The Planning Phase

Entry Criteria: approved Project Charter and designation of a project champion and a project manager; identified key stakeholders and a team of domain experts.

Phase Focus and Needs: The project champion's main role is to act as the liaison between the upper management and the project team including the project manager, so as to ensure that the project moves smoothly from its planning phase to its execution phase.

The project manager must define the project's vision, objectives, and business case. Much of the planning, estimation, and scheduling takes place while defining the basic set of requirements (Figure 3).

Engineering Process: Unified Process (UP) is suitable for this phase. It enables the manager to focus on planning activities necessary for sound project management. The inception phase of UP combines key efforts of the formal project planning tasks and software engineering kick-off. It is known to produce a solid set of planning documents that are necessary for formal project management (Figure 3) (Ambler, 2002).

Exit Criteria: a set of project planning documents including project charter, business case, integrated master schedule, risk register, use-case models, and initial requirements.

The Innovation Phase

Entry Criteria: high level of uncertainty; only key stakeholders are involved; basic set of requirements.

Phase Focus and Needs: During the early stage of executing a project, the needs and focus of the engineering process centers on producing a working software based on a basic set of requirements while resolving issues relating to the business domain and new technologies. Giving the team a higher level of autonomy and freedom will empower it to find innovative solutions that address much of the unknown aspects of development early-on. The project manager must select an approach that focuses on construction (Figure 3).

Engineering Process: Extreme Programming (XP) is most suitable for this phase because it fosters innovation and fast-paced programming. It enables both customer and developer to deal with most uncertainties up front in a dynamic way. XP is known to be capable of producing working software early-on (Figure 3) (Beck, 1999).

Exit Criteria: a working product that encompasses resolution of most issues relating to the business domain and new technology.

The Organization Phase

Entry Criteria: a working product and expanded set of requirements; most stakeholders identified.

Phase Focus and Needs: bringing order to the chaos affected by the innovation phase. The working software must be refined to address specific needs that may be uncovered by working with additional stakeholders. As the project progresses, uncertainty decreases and the development effort must be organized to ensure all necessary aspects of the software are properly addressed. To achieve this, the project manager must select an approach that balances construction, with analysis and testing, and design documentation (Figure 3).

Engineering Process: Scrum cuts through project complexity and chaos by enabling a team to organize itself around a defined work routine, which allows a particularly productive order to emerge. It organizes the development process around a prioritized list according to the customer's requirements. It also serves as a safety net against the calendar by enabling both customer and developer to declare the project done any time if needed (Figure 3) (Schwaber, 2004).

Exit Criteria: working software that is well tested with specific features based on the prioritized list of requirements; testing results; and initial design documentation.

The Definition Phase

Entry Criteria: a working product that can be declared done; all stakeholders have been identified; complete set of requirements.

Phase Focus and Needs: produce complete set of design documentation and work out last minute issues. The final version of the software must be delivered according to the transition plan. Uncertainty at this stage is brought down to a minimum and the software project can be fully defined in no uncertain terms. The project manager must select an approach characterized by high discipline and formal documentation (Figure 3).

Engineering Process: Waterfall is capable of producing solid design documentation necessary for future development. It brings a higher level of discipline at the end of the development process to ensure orderly delivery and transition to the customer. It may also continue after the project declared done in the form of updates, upgrades, and design documentation (Figure 3) (Royce, 1970).

Exit Criteria: finished software product; and complete design documentation.

The Closing Phase

Explaining issues relating to this phase is outside the scope of this framework. It is mentioned here to assert its importance as part of the project management process. Additional research may uncover effective approaches that can be integrated into the framework.

Linkage between phases

Within the context of the framework, transitioning from one executing phase to another is a process of mapping work products from a phase as assets for the subsequent one. Additional research and work is needed to establish the technical details of such mappings. However, it is useful to shed some light on the linkage between the phases and the issues that must be taken into consideration.

Division between phases provides a convenient means to show stages of approvals, reviews, documentation, and other milestones of the project. Following the end of one phase, the plan for the next phase can be developed in earnestness and in detail. It is a convenient way to think of the project as progressing from one phase to the next.

At the same time, some blurring of activities can occur across phase boundaries. Some of the activities, though spanning over different phases, may nevertheless be interdependent. More importantly, not all activities in a project are strictly sequential in nature, and feedback and rework of activities are normal and healthy, as additional details are uncovered. For example, although the project plan is an outcome of the planning phase, the plan may change in subsequent phases as new information becomes available.

Another linkage between the phases has to do with the degree of impact on decisions made and errors committed. It is a well-known principle often quoted in software engineering that the effect of a decision on the project's final outcome is high at the planning and innovation phases, and less if made in the later phases. Similarly the cost impact of errors made is very high during the early stages, and reduces over subsequent phases.

Another area of relative difference is the effect or cost of changes across the phases. With each succeeding phase that the project has entered, the cost of changes increases. All of these observations are supported by case projects, but are difficult to prove across all software projects in an objective way.

Project management practices may differ from one phase to another because different activities are performed in the different phases. Similarly, different project management skills may be brought to play to effectively succeed in accomplishing the tasks required in the different phases. Of course, some of the same project management activities may occur in all the phases, such as stakeholder involvement, change management, negotiation, and measurement of resources utilized and progress made.

The main outputs of the planning phase are documents, and these may be seen by owners as not adding sufficient value or not worth the investment made to produce

them. This may result in not adequately performing the planning tasks. Certainly, owners can and should be educated to the reality that if early work is not done well, then project outcomes will almost certainly be undesirable, and one can cite plenty of examples of software projects that have failed when early phase work was not carried out or not done properly.

Time Distribution between Phases

Uncertainty in the outcome of a software project occurs due to various factors such as project complexity, definitiveness of requirements, stakeholders' involvement, and uniqueness of the project with respect to its business and technical environments. However, regardless of its level, uncertainty does not remain the same throughout the project life span. It changes from high at the beginning of the project, to moderate in the middle, to low towards the end, as more factors are mitigated or uncovered (Schwalbe, 2006). Projects with higher levels of uncertainty require more innovation and organization, so less time is dedicated to definition. On the other hand, projects with lower level of uncertainty require less innovation and organization, so more time can be dedicated to definition (Figure 4).

Additional research is needed to determine whether a defined relationship exists between the level of uncertainty and the percentage of time dedicated to each phase. For example, it will be nice to have a formula or at least some heuristic way to estimate the overall level of uncertainty for a software project based on the following key factors:

- Stakeholders involvement
- Definitiveness of requirements
- Project complexity
- Technology newness
- Team competence

Another formula or heuristic would be useful to determine the percentage of time that should be allocated to each of the executing phases: innovation, organization, and definition.

5. BENEFITS OF THE DISCIPLINED AGILITY PROCESS FRAMEWORK

The benefits of the framework from the project management perspective are as follows:

- Clarity of the engineering process and its ability to deal with change. The framework emphasizes tackling issues with high level of uncertainty early on. Delivering the product in stages reduces the technical risk of unsuccessful integration or inadequate testing. The use of Extreme Programming and Scrum reduces the risk associated with changing requirements.
- Control project calendar with the ability to distribute time between phases based on the overall uncertainty level of the project. Time can also be redistributed to executing phases based on progress. The end of each phase presents tangible signs of progress and opportunities to revise plans. It reduces the risk of slipping behind schedule.
- Control cost with the ability to drop less important requirements and shorten the definition phase. If the project runs the risk of budget overrun, the prioritized requirements list allows the manager to postpone less important requirements. The manager can also control the amount of time spent on the design documentation during the definition phase without jeopardizing the final delivery of the project.
- Discipline with sharp milestones at the end of each phase and formal work products at the beginning and end of the project. The framework defines key milestones with entry and exit criteria for each phase that focus on the software product.

The benefits of the framework from the software engineering perspective are as follows:

- Structured approach with defined entry/exit criteria and time frame for each phase. The team knows what is expected and the process to follow. The progressive approach towards a higher level of discipline allows the

team to adjust gradually as the project moves forward.

- Flexibility to deal with changing requirements and priorities during the first two phases of the project because final design decisions will be made at a later phase. The team does not have to worry about major changes to requirements on ongoing basis because they know it will be considered periodically at the end of each phase.
- Freedom to innovate early in the project when it is most needed by focusing immediately on construction and working directly with code. Problems and shortcomings surface early and the team can focus on finding novel ways to resolve them.

The benefits of the framework from the sponsors' and customers' perspective are as follows:

- Immediate results that propel better buy in. The framework is designed to deliver the most relevant functionality first. Users don't have to wait for a particular functionality until the full product is ready.
- Satisfaction with features and quality. The prioritized requirements list assist in providing the customers with the features they want. The staged delivery ensures shorter feedback cycle and improves quality.
- Clarity in the design to enable future development. Software products that lack design documents are difficult to modify in the future and risk being outdated and inadequate for customers' changing needs.

6. SUMMARY AND CONCLUSION

In order for software projects to succeed, it is critical to apply the appropriate planning methods such as developing an integrated master schedule that identifies and logically links all project milestones and critical tasks, assigning budget and resources to tasks, and monitoring progress throughout the life of the project. In addition, a risk register should be created to capture and minimize risks associated with the project. However, there's a need for a software engineering framework that enables the project manager to harness the best of known engineering processes in an agile, but disciplined, manner. Agility is important to deal with change and uncertainty, and discipline is important to establish plans, manage deviations, and meet responsibilities and constraints. The Disciplined Agility Framework addresses three key issues for the project manager. First, it provides a workable balance between the need for certainty and the need for agility in software project management. Second, it defines a software engineering process that is effective regardless of the characteristics of the project. And third, it addresses many of the needs of the key stakeholders, namely the project manager, the engineering team, the sponsors, and most importantly, the customer, as they work through the project's life cycle.

In this paper we first provided an overview of Software Project Management and Software Engineering Processes and highlighted the difficulties that often lead to project failure. We then showed why known engineering processes are less than effective in addressing software project needs. Much of the existing literature on software engineering processes stress the need for approaches that combine advantages from traditional and agile methods, but do not offer a practical way to do so. In this paper we proposed a framework that enables the project manager to harness the best of known engineering processes in an agile, but disciplined, manner. The concept is based on the evolutionary nature of software projects. Plans, requirements, stakeholders' involvement, and uncertainty change over their life span. The framework divides the project into five phases: planning, innovation, organization, definition, and closure. We examined each phase in detail. Finally, we presented a discussion of the benefits of the framework from the project management perspective, the software engineering perspective, and the sponsors/customers perspective.

7. ACKNOWLEDGMENT

The Disciplined Agility Framework is the work of Adam Noureddine. The authors gratefully acknowledge the feedback and suggestions from the referees.

8. REFERENCES

Ambler, S., and L. Constantine (2002) The Unified Process Inception Phase, CMP Books.

Beck, K. (1999) Extreme Programming Explained: Embrace Change. Addison-Wesley.

Brooks, F. (1975) The Mythical Man-Month, Addison-Wesley.

Henderson, Peter (2006) "Why Large IT Projects Fail," Retrieved from: http://de.scientificcommons.org/43269531

McConnell, Steve (1998) Software Project Survival Guide, Microsoft Press.

Pressman, R. (2005) Software Engineering, A Practitioner's Approach, McGraw Hill.

Project Management Institute, Inc. (2004) A Guide to the Project Management Body of Knowledge (PMBOK® Guide).

Royce, W. W. (1970) "Managing the Development of Large Software Systems: Concepts and Techniques," Proc. WESCON.

Schwaber, Ken (2004) Agile Project Management with Scrum, Microsoft Press.

Schwalbe, Kathy (2006) Information Technology Project Management, Fourth Edition, Thomson Course Technology.

The Standish Group (1995) "The CHAOS Report" Retrieved from: www.standishgroup.com

The Standish Group (2009) "CHAOS Summary" Retrieved from: www1.standishgroup.com/newsroom/chaos_2009.php

APPENDIX

<table>
<tr><td rowspan="2"></td><td colspan="4">Project Life Span</td></tr>
<tr><td rowspan="2">Project Management Tasks</td><td>Planning Phase</td><td colspan="2">Executing Phase</td><td>Closing Phase</td></tr>
<tr><td rowspan="2">Initiate Project</td><td>Gather more Requirements</td><td rowspan="2">Deliver Product</td><td rowspan="2">Project Closure</td></tr>
<tr><td>Refine Plans</td></tr>
</table>

Figure 1: Project management phases

	Executing Phase		
Uncertainty Progress	High	Moderate	Low
Stakeholders Involved	Key	Most	All
Requirements Set	Basic	Prioritized	Elaborated
Project Needs	Innovation	Organization	Definition

Figure 2: Sub-phases based on project needs during execution

<table>
<tr><th></th><th rowspan="2">Planning Phase</th><th colspan="3">Executing Sub-Phases</th></tr>
<tr><th>Innovation</th><th>Organization</th><th>Definition</th></tr>
<tr><td rowspan="2">Development Process</td><td>UP</td><td>XP</td><td>Scrum</td><td>Waterfall</td></tr>
<tr><td>Provides excellent clarity during the initiation phase.</td><td>Enables both customer and developer to deal with most uncertainties up front with the ability to produce a working software early-on.</td><td>Serves as a safety net and enables both customer and developer to declare the project done if needed.</td><td>Necessary for future development and may continue after the project declared done in the form of updates and design documentation.</td></tr>
<tr><td rowspan="4">Development Focus</td><td rowspan="4">Planning</td><td rowspan="2">Construction</td><td rowspan="2">Construction</td><td>Construction</td></tr>
<tr><td>Analysis</td></tr>
<tr><td rowspan="2">Analysis/Testing</td><td rowspan="3">Design - Documentation</td></tr>
<tr><td>Analysis</td><td rowspan="2">Design - Documentation</td></tr>
<tr><td>Design - Documentation</td></tr>
<tr><td>Software Product</td><td>Project Plan</td><td>Working Set</td><td>Specific Features</td><td>Completed Set</td></tr>
</table>

Figure 3: Balancing analysis, design, construction, testing and documentation during the executing phase

<table>
<tr><th></th><th colspan="4">Phase Time Distritbution
(Overall uncertainty level defines time distribution for the execution phases)</th></tr>
<tr><td>Low Uncertainty</td><td>Initiation
Unified Process</td><td>Innovation
XP</td><td>Organization
Scrum</td><td>Definition
Waterfall</td></tr>
<tr><td>Moderate Uncertainty</td><td>Initiation
Unified Process</td><td>Innovation
XP</td><td>Organization
Scrum</td><td>Definition
Waterfall</td></tr>
<tr><td>High Uncertainty</td><td>Initiation
Unified Process</td><td colspan="2">Innovation
XP</td><td>Organization
Scrum</td><td>Definition
Waterfall</td></tr>
</table>

Figure 4: Time distribution between phases based on the uncertainty level