

**CHALMERS**



**UNIVERSITY OF GOTHENBURG**

# Measuring Agility

*Master's Thesis in Software Engineering*

KONSTANTINOS CHRONIS

Division of Software Engineering  
Department of Computer Science & Engineering  
GOTHENBURG UNIVERSITY  
CHALMERS UNIVERSITY OF TECHNOLOGY  
Gothenburg, Sweden 2014  
Master's Thesis 2014:6



## Abstract

**Context:**

**Objective:**

**Method:**

**Results:**



## **Acknowledgements**

Konstantinos Chronis, Gothenburb, Sweden June 1, 2014



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Thesis Structure . . . . .	2
<b>2</b>	<b>Related Work</b>	<b>3</b>
2.1	Introduction . . . . .	3
2.2	Agility of Methodologies . . . . .	3
2.2.1	Balancing Discipline and Agility . . . . .	3
2.2.2	Philip Taylor - Assessing Tool . . . . .	4
2.2.3	Datta - Agility Measurement Index . . . . .	4
2.2.4	Comprehensive Evaluation Framework for Agile Methodologies . .	5
2.2.5	XP Evaluation Framework . . . . .	5
2.2.6	4-Dimensional Analytical Tool . . . . .	6
2.3	Agility of Organisations . . . . .	7
2.3.1	Team Agility Assessment . . . . .	7
2.3.2	Comparative Agility . . . . .	8
2.3.3	Thoughtworks . . . . .	8
2.3.4	Perceptive Agile Measurement . . . . .	8
2.3.5	Shawky and Ali - Entropy Analysis . . . . .	8
2.3.6	Escobar - Vasquez Model for Assessing Agility . . . . .	8
2.3.7	Validation Model to Measure the Agility . . . . .	9
2.3.8	AHP - ANFIS Framework . . . . .	10
2.3.9	42-Point Test . . . . .	11
2.3.10	Sidky Agile Measurement Index . . . . .	11
2.3.11	OPS Framework . . . . .	12
<b>3</b>	<b>Case Study</b>	<b>15</b>
3.1	Problem . . . . .	15
3.1.1	Research Questions . . . . .	15
3.2	Method . . . . .	15
3.2.1	Case Study Design . . . . .	15

3.2.2	Case and Subjects Selection . . . . .	15
3.2.3	Observations . . . . .	15
3.2.4	Company F . . . . .	16
3.2.5	Methodology F . . . . .	16
3.2.6	Teams . . . . .	16
3.2.7	Products . . . . .	17
3.3	OPS . . . . .	18
3.3.1	Introduction . . . . .	18
3.3.2	Adequacy Assessment . . . . .	18
3.3.3	Capability and Effectiveness Assesement . . . . .	20
3.3.4	Results Verification . . . . .	27
3.4	Perceptive Agile Measurement . . . . .	27
3.4.1	Results Verification . . . . .	28
3.5	Team Agility Assessment . . . . .	28
3.5.1	Results Verification . . . . .	28
3.6	Thoughtworks . . . . .	28
3.6.1	Results Verification . . . . .	28
3.6.2	OPS . . . . .	28
3.6.3	PAM . . . . .	28
3.6.4	Leffingwell . . . . .	28
3.7	Discussion . . . . .	28
<b>4</b>	<b>Tools Completeness</b>	<b>29</b>
4.1	Team Agility Assessment Areas . . . . .	29
4.2	Perceptive Agile Measurement Practices . . . . .	30
4.3	Objectives, Principles, Strategies Practices . . . . .	30
4.4	Practices Covered Between The Tools . . . . .	31
4.5	Mapping of questions from the PAM and TAA tools . . . . .	33
4.5.1	Questions to Practices/Strategies . . . . .	34
4.5.2	Non Relevant Questions . . . . .	39
4.6	Results . . . . .	40
	<b>Appendices</b>	<b>41</b>
<b>A</b>	<b>The Capability and Effectiveness Hierarchy for Company F</b>	<b>42</b>
A.1	Capability Hierarchy . . . . .	42
A.2	Effectiveness Hierarchy . . . . .	46
<b>B</b>	<b>Effectiveness Questions for Company F's Teams</b>	<b>51</b>
<b>C</b>	<b>Perceptive Agile Measurement</b>	<b>53</b>
<b>D</b>	<b>Team Agility Assessment</b>	<b>56</b>



**Bibliography**

**62**

# List of Tables

2.1	Levels of software method understanding and use (after Cockburn) . . . .	4
2.2	4-DAT Dimension 1 . . . . .	6
2.3	4-DAT Dimension 2 . . . . .	7
2.4	4-DAT Dimension 3 . . . . .	7
2.5	4-DAT Dimension 4 . . . . .	7
2.6	AHP - ANFIS Framework parameters . . . . .	11
3.1	Practices embraced in methodology F . . . . .	16
3.2	Team A - Profile . . . . .	17
3.3	Team B - Profile . . . . .	17
3.4	Team C - Profile . . . . .	17
3.5	Team D - Profile . . . . .	17
3.6	Scale for Capability . . . . .	21
3.7	Scale for Effectiveness . . . . .	21
3.8	Scale for intrepreting final scores . . . . .	21
4.1	Areas covered by TAA . . . . .	29
4.2	Agile practices covered by PAM . . . . .	30
4.3	Agile practices covered by OPP . . . . .	31
4.4	Relation of OPP/OPS and PAM practices . . . . .	32
4.5	Relation of OPP/OPS and TAA practices/areas . . . . .	33
4.6	Questions Coverage from OPS . . . . .	40

# List of Figures

2.1	Dimensions affecting method selection . . . . .	4
2.2	Evaluation criteria hierarchy for CEFAM . . . . .	5
2.3	Escobar - Vasquez model for assessing agility . . . . .	9
2.4	Validation Model to Measure the Agility . . . . .	10
2.5	Objectives, Principles, and Strategies identified by the OPS Framework .	13
2.6	OPS Framework . . . . .	14
3.1	Objectives identified in methodology F . . . . .	18
3.2	Principles identified in methodology F . . . . .	19
3.3	Strategies identified in methodology F . . . . .	20
3.4	Capability - Strategies level . . . . .	25
3.5	Capability - Principles level . . . . .	25
3.6	Capability - Objectives level . . . . .	26
3.7	Effectiveness - Strategies level . . . . .	26
3.8	Effectiveness - Principles level . . . . .	27
3.9	Effectiveness - Objectives level . . . . .	27

# 1

## Introduction

**A**GILE and plan-driven methodologies are the two dominant approaches in the software development. The eternal battle between them seems will go on for a long time before one of them prevails. Organizations and companies have the tendency to leave the cumbersome area of Waterfall process and to embrace the Agile methodologies. Although it has been almost 20 years since the latter were introduced, companies are quite reluctant in following them. Once they do, they start enjoying the agile benefits, but are these the only benefits they could enjoy?

In order to answer to the previous question one should first understand what does “agile” mean? According to a dictionary, it means “to be able to move quickly and easily”, something which is almost impossible with a plan-driven approach. The term agility was first introduced as agile manufacturing in an industry book [21].

In 2001, 17 developers formed the Agile Alliance and created the agile manifesto [7] defining what is considered to be agile in order to avoid confusion:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Software development teams started to adopt the most known agile methodologies, such as eXtreme Programming [6], Feature Driven Development (FDD) [23], Crystal [11], Scrum [29] and others. Most companies use a tailored methodology by following some processes and practices of the aforementioned ones in order to better suit their needs. Williams et al. [42] reports that rarely all XP practices are exercised in their pure form something to which Reifer [27] and [5] also agree based on the results of their surveys showing that it is common for organizations to partially adopt XP. Sidky et al.

[32] mention four issues organisations face when transitioning to agile. a) the organization's readiness for agility b) the practices it should adopt c) the potential difficulties in adopting them d) the necessary organizational preparations for the adoption of agile practices. The most important issue though that tends to be neglected, is how well are these methodologies adopted?

According to Escobar-Sarmiento and Linares-Vasquez [13] the agile methodologies are easier to misunderstand. Such a case could lead to problems later on in the software development process. The aforementioned statement is also supported by Taromirad and Ramsin [38] as well who argue that the agile software development methodologies are often applied to the wrong context. Hossain et al. [15] argues that improper use of agile practices creates problems. Sahota [28] states that doing agile and being agile are two different things. For the first one a company should follow practices while for the other one a company should think in an agile way. Lappo and Andrew [19] state that organizations following the practices of a methodology does not mean they gain much in terms of agility, while on the other hand Sidky [31] defines the level of agility of a company as the amount of agile practices used. Considering the aforementioned statement, a group that uses pair programming and collective code ownership at a very low level is more agile than a group which uses only pair programming but in a more efficient manner.

Williams et al. [43] pose the question “How agile is agile enough”? Practitioners think that declaring being agile is equally good as being agile. According to a survey from Ambyssoft [3] only 65% of the agile companies that answered met the five agile criteria posed in the survey. In addition, 9% of agile projects failed due to lack of cultural transition while 13% of companies are at odds with core agile values based on the most recent survey by VersionOne [40]. Poonacha and Bhattacharya [24] mentioned that the different perception of agile practices when they are adopted is very worrying, since even people in the same team understand them differently according to the result of a survey [2]. It is evident not only from literature but also from its application that agile is a way of thinking and working, it is a culture [24]. If we had to use one word we could state it is a way of *being*. Nietzsche [22] said “better know nothing than half-know many things”. In the same context maybe it is better not to transition to agile instead of thinking being agile.

Since agile methodologies become more and more popular there is a great need for development of a tool that can measure the level of agility in the organizations that have adopted them. Researchers for more than a decade have been constantly coming up with models and frameworks in an effort to provide a solution. Unfortunately the multiple tools have created a saturation in the field resulting in being used only from the organizations that participated in the empirical studies for their creation [17]. As a result, the vicious circle of creating tools with no actual use does not stop holding back not only the software development companies, but the research community as well.

## 1.1 Thesis Structure

# 2

## Related Work

### 2.1 Introduction

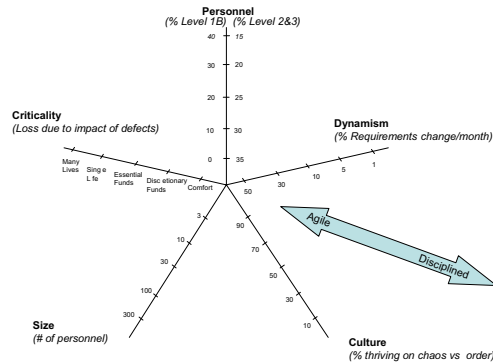
VARIOUS TOOLS have been developed in the last decade in order to measure the agility in software development teams. Below is a short description of some of the ones that have been used as references in papers of this field. The tools are separated in two categories, a) those which measure the agility of the methodologies and b) those which measure the agility of organisations/teams.

### 2.2 Agility of Methodologies

#### 2.2.1 Balancing Discipline and Agility

Boehm and Turner [8] did not come up with a tool to measure agility but rather to balance between agility and discipline. According to them [9] discipline is the foundation for any successful endeavor and creates experience, history and well-organized memories. On the other hand agility is described as a counterpart of discipline. Agility uses the memory and history in order to adjust in the context which is applied and takes advantage of the unexpected opportunities that might come up. The combination of the two can bring success to an organisation. Boehm and Turner [8] in their research found that there are five “critical decision factors” which can determine if an agile or plan-driven method is suitable for a software development project.

Figure 2.1 depicts these factors: a) size of a team working in a project b) criticality of damage of unexpected defects c) culture needed to balance between chaos and order d) dynamism of the team working in chaos or in a planned way e) personnel which refers to the extended Cockburn [10] skill rating



**Figure 2.1:** Dimensions affecting method selection

Level	Characteristics
3	Able to revise a method (break its rules) to fit an unprecedented newsituation
2	Able to tailor a method to fit a preecedented new situation
1A	With training, able to perform discretionary method steps (e.g., sizing stories to fit increments, composing patterns, compound refactoring, complex COTS integration). With experience can become Level 2.
1B	With training, able to perform procedural method steps (e.g. coding a simple method, simple refactoring, following coding standards and CM procedures, running tests). With experience can master some Level 1A skills.
-1	May have technical skills, but unable or unwilling to collaborate or follow shared methods.

**Table 2.1:** Levels of software method understanding and use (after Cockburn)

If the ratings of the five factors are close to the center, then the team is to an agile territory and the team is considered agile, otherwise it follows a discipline approach.

### 2.2.2 Philip Taylor - Assessing Tool

Taylor et al. [39] modified the tool created by Boehm and Turner [8] by adding a sixth axis named *Client Involvement* which has the following categories:

- On AB - Client is on-site and an agile believer. This is the ideal when a client is fully persuaded of the agile approach and makes themselves available onsite to work with the team.
- Off AB - Client is off-site but an agile believer. Although off-site, the client fully understands the nature of agile development and is open to frequent communication.
- On AS - Client is on-site but is an agile skeptic. They may be on-site but they are not convinced about the agile development approach.
- Off AS - Same as On AS except the problem is compounded by the client being off-site.
- Off Uninvolved - Not only is the client off-site but they want no involvement between providing the initial requirements and getting the right product delivered.

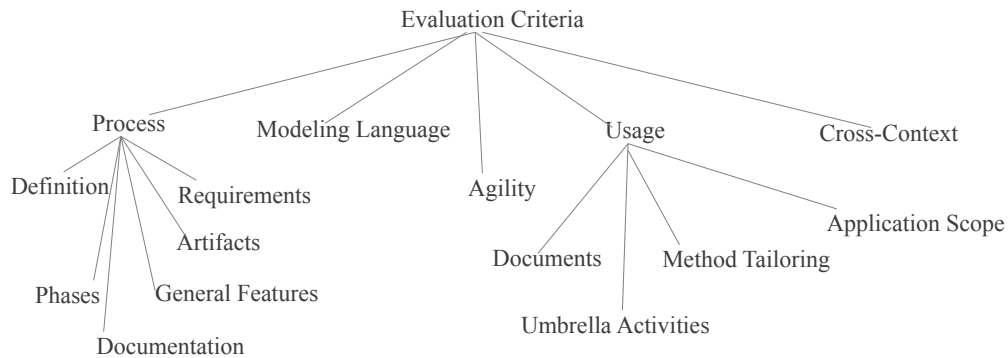
### 2.2.3 Datta - Agility Measurement Index

Datta [12] presented a metric to help in deciding which agile methodology best suits a project. The metric identifies five dimensions: a) Duration b) Risk c) Novelty d) Effort

e) Interaction. For each one of these dimensions the user assigns a value, then by the use of a formula the user can identify whether Waterfall, Unified Process or eXtreme Programming is more appropriate.

#### 2.2.4 Comprehensive Evaluation Framework for Agile Methodologies

Taromirad and Ramsin [38] created the “Comprehensive Evaluation Framework for Agile Methodologies” (CEFAM) in order to provide coverage to the important aspects of agile methodology. The tool consists of a hierarchy of evaluation criteria which are divided into five groups (see Figure 2.2) a) Process b) Modeling Language c) Agility d) Usage e) Cross-Context. Each of these groups has a number of questions which are either answered with a numeric value, with Yes/No or any value from a proposed set. At the end the answers are evaluated based on the following scale: Unacceptable  $\leq 0.25$ ;  $0.25 < \text{Low} \leq 0.5$ ;  $0.5 < \text{Medium} \leq 0.75$ ;  $0.75 < \text{High} \leq 1.0$ .



**Figure 2.2:** Evaluation criteria hierarchy for CEFAM

#### 2.2.5 XP Evaluation Framework

Williams et al. [42] proposed a framework named the “XP Evaluation Framework” (XP-EF) for assessing the XP practices which have been adopted by an organization. The framework consists of three parts

- XP Context Factors (XP-CF) - Record a important contextual information. The factors can be team size, project size, staff experience
- XP Adherence Metrics (XP-AM) - Epxress in a precise way the practices utilized by a team
- XP Outcome Measures (XP-OM) - A Means to assess the outcomes of a project using full or partial XP practices



### 2.2.6 4-Dimensional Analytical Tool

Qumer and Henderson-Sellers [25] created the 4-Dimensional Analytical Tool (4-DAT) for analysing and comparing agile methods which is part of the Agile Adoption and Improvement Model (AAIM) [26]. The objective of the tool is to provide a mechanism to assess the degree of agility and adoptability of any agile methodology. The measurements are taken at a specific level in a process and using specific practices.

**Dimension 1 - Method Scope Characterization** The first dimension describes the key scope items which have been derived from their literature review based on Beck and Andres [6], Koch [18], Palmer and Felsing [23], Highsmith [14] and provides a method comparison at a high level.

These items are: a) Project Size b) Team Size c) Development Style d) Code Style e) Technology Environment f) Physical Environment g) Business Culture h) Abstraction Mechanism

The aforementioned elements are considered essential for supporting the method used by a team or organisation. Table 2.2 provides a description for the items.

Scope	Description
1. Project Size	Does the method specify support for small, medium or large projects (business or other)?
2. Team Size	Does the method support for small or large teams (single or multiple teams)?
3. Development Style	Which development style (iterative, rapid) does the method cover?
4. Code Style	Does the method specify code style (simple or complex)?
5. Technology Environment	Which technology environment (tools, compilers) does the method specify?
6. Physical Environment	Which physical environment (co-located or distributed) does the method specify?
7. Business Culture	What type of business culture (collaborative, cooperative or non-collaborative) does the method specify?
8. Abstraction Mechanism	Does the method specify abstraction mechanism (object-oriented, agent-oriented)?

Table 2.2: 4-DAT Dimension 1

**Dimension 2 - Agility Characterization** The second dimension is the only quantitative dimension of the four. It evaluates the agile methods in process level and in a method practices level in order to check the existence of agility.

The measurement of the degree of agility in this level is done based on the following five variables. Table 2.3 provides a description for them. a) Flexibility b) Speed c) Leanness d) Learning e) Responsiveness

These variables are used to check the existence of a method's objective at a specific level or phase. If the variable exists for a phase then the value 1 is assigned to it, otherwise 0. Qumer and Henderson-Sellers [25] define the degree of agility (DA) as "the fraction of the five agility variables that are encompassed and supported".

The function for calculating the DA is the following

$$DA(Object) = (1/m) \sum m DA(Object, PhaseOrPractices) \quad (2.1)$$

Features	Description
1. Flexibility	Does the method accommodate expected or unexpected changes?
2. Speed	Does the method produce results quickly?
3. Leanness	Does the method follow shortest time span, use economical, simple and quality instruments for production?
4. Learning	Does the method apply updated prior knowledge and experience to learn?
5. Responsiveness	Does the method exhibit sensitiveness?

Table 2.3: 4-DAT Dimension 2

**3 - Agile Values Characterization** The third dimension consists of six agile values. Four of them are derived directly from the Agile Manifesto [7], while the fifth comes from [18]. The last value is suggested by Qumer and Henderson-Sellers [25] after having studied several agile methods. Table 2.5 shows the agile values.

Agile values	Description
1. Individuals and interactions over processes and tools	Which practices value people and interaction over processes and tools?
2. Working software over comprehensive documentation	Which practices value working software over comprehensive documentation?
3. Customer collaboration over contract negotiation	Which practices value customer collaboration over contract negotiation?
4. Responding to change over following a plan	Which practices value responding to change over following a plan?
5. Keeping the process agile	Which practices helps in keeping the process agile?
6. Keeping the process cost effective	Which practices helps in keeping the process cost effective?

Table 2.4: 4-DAT Dimension 3

**Dimension 4 - Software Process Characterization** The fourth dimension examines the practices that support four processes as these are presented by Qumer and Henderson-Sellers [25]. Table 2.5 lists these processes.

Process	Description
1. Development Process	Which practices cover the main life cycle process and testing (Quality Assurance)?
2. Project Management Process	Which practices cover the overall management of the project?
3. Software Configuration Control Process / Support Process	Which practices cover the process that enables configuration management?
4. Process Management Process	Which practices cover the process that is required to manage the process itself?

Table 2.5: 4-DAT Dimension 4

## 2.3 Agility of Organisations

### 2.3.1 Team Agility Assessment

Leffingwell [20] created a model for assessing the team's agility by taking into account six aspects: a) Product Ownership b) Release Planning and Tracking c) Iteration Planning and Tracking d) Team e) Testing Practices f) Development Practices/Infrastructure

Each of these aspects is followed by a number of questions rated in a Likert scale 0-5 and the results are represented in a radar chart.

### 2.3.2 Comparative Agility

Williams et al. [43] created the Comparative Agility (CA) assessment tool which does not assess the agility of an organization by providing an absolute value, but it rather provides a value in comparison to other organizations/companies [1]. The idea behind CA is that organizations try to be more agile than their competitors because of believing to have more benefits. Until 2010 more than 1200 respondents supported that idea by answering the tool's online survey. The CA assessment tool consists of the following seven dimensions a) Teamwork b) Requirements c) Planning d) Technical Practices e) Quality f) Culture g) Knowledge-Creating which are made up of three to six characteristics each one of them. Each characteristic has four statements where each one of them represents an agile practice. The answers of every statement are on five-point Likert scale.

### 2.3.3 Thoughtworks

Thoughtworks [36] is a worldwide consulting company. They have developed an online survey for assessing agility. People can answer to the survey and they will get a report evaluating at which level their team or company is.

### 2.3.4 Perceptive Agile Measurement

So and Scholl [33] from a social-psychological perspective created a survey for measuring agility covering eight agile practices which they named as scales. These scales are an attempt to establish a representative set of agile practices commonly used in the field

a) Iteration Planning b) Iterative Development c) Continuous Integration and Testing d) Co-Location e) Stand-up Meetings f) Customer Access g) Customer Acceptance Tests h) Retrospectives.

### 2.3.5 Shawky and Ali - Entropy Analysis

Shawky and Ali [30] measure the agility based on the rate of entropy change over the time of a system's development. If the rate is high then the process is of high agility as well. Each feature is considered to be an entity and the change logs of the entities are analyzed. They define as  $P_i(t)$  the probability of an entity  $i$  to be associated with the change logs at a time  $t$ . Then by using the following formula they make the calculation for the agility measure  $AM(t)$  for that specific time.

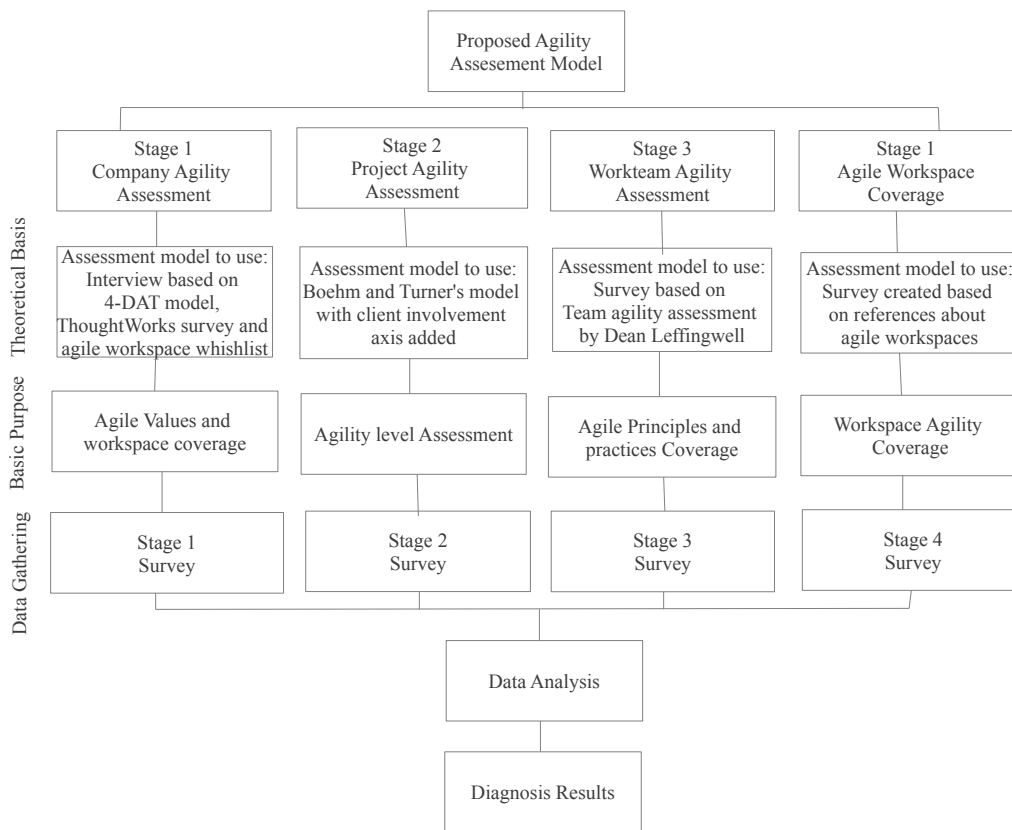
$$AM(t) = - \sum_{i=1}^n P_i'(t)(\log_2 P_i(t) + 1.44) \quad (2.2)$$

### 2.3.6 Escobar - Vasquez Model for Assessing Agility

Escobar-Sarmiento and Linares-Vasquez [13] created their own agility assessment model which consists of four stages. For the first three they use the models and tools proposed by other researchers they found in literature.

- Agile Project Management Assessment - proposed by Qumer and Henderson-Sellers [25]
- Project Agility Assessment - proposed by Taylor et al. [39]
- Workteam Agility Assessment - proposed by Leffingwell [20]
- Agile Workspace Coverage

For collecting the data for the measurements they used surveys based on the tools of each stage while in the last one they use their custom survey. The data are then depicted in a four axis radar chart in order to provide a view of the company's agility. In Figure 2.3 one can see the model with a short description about which tool should be used at each level for each stage.



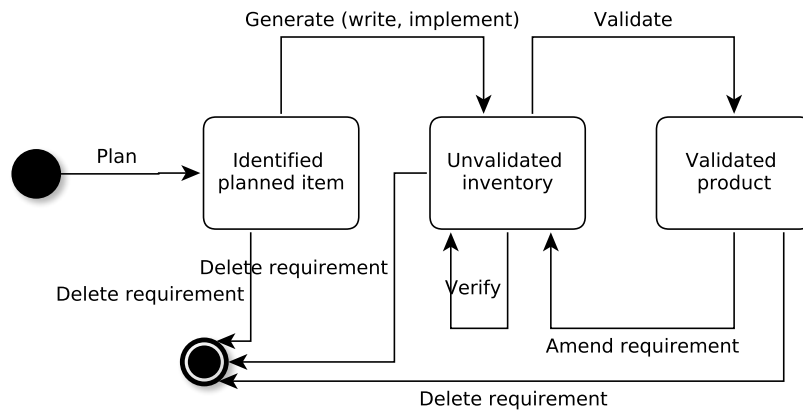
**Figure 2.3:** Escobar - Vasquez model for assessing agility

### 2.3.7 Validation Model to Measure the Agility

Ikoma et al. [16] measure agility by creating a validation model since according to them only validation can confirm the quality of a product. In this model any candidate for

validation item enters an “identified planning state” at planning time. Afterwards these items change to the “unvalidated inventory state” when the items start to be generated. Finally, validation of the deliverable items changes the state to the “validated product state” (see Figure 2.4). Then based on the following equation once can get the result.  $A$  is the agility of a project/organization,  $V'$  is the number of software items in the “validated product state” and  $U'$  is the average number of software items in which intermediate deliverables are in the “unvalidated inventory state”.

$$A = V'/U' \quad (2.3)$$



**Figure 2.4:** Validation Model to Measure the Agility

### 2.3.8 AHP - ANFIS Framework

Poonacha and Bhattacharya [24] created tool for measuring agility by identifying 17 parameters grouped in four parameter groups as it can be seen in Table 2.6. While the last group is an indicator of performance, the first three groups mitigate the risks of supply, operation and demand uncertainties respectively. Each parameter is given as a question and the answers were fed in the Adaptive Network based Fuzzy Inference Systems (ANFIS). Due to the complexity of the ANFIS model an Analytical Hierarchical Process (AHP) is mandatory to minimize it.

Group	Parameters
People	a) Attrition b) Functional Flexibility c) Training and Knowlegde d) Decentralized Decision Making e) Bench Strength
Processes	a) Pair Programming and Parallel Testing b) Iterative Development c) Degree of modularity d) Requirement Capture Process e) Reusability f) Continuous Improvement
Customer Involvement	a) Customer Involvement in Design b) Team Across Company Borders c) Customer Training Period
Cost and Quality	a) Cost of Requirement change b) Projects dropped due to incapacity c) Software Quality

**Table 2.6:** AHP - ANFIS Framework parameters

### 2.3.9 42-Point Test

Waters [41] developed createad a simple 42-question survey based on a similar one from Nokia [37], in order to allow to Scrum/XP teams to easily see to what extent they follow various agile practices.

### 2.3.10 Sidky Agile Measurement Index

Sidky [31] created the Sidky Agile Measurement Index (SAMI) in order to measure agility as part of the “Agile Adoption Framework”. SAMI is a scale used by an agile coach to identify the potential of a project or organization [32], which consists of five agile levels and five agile principles, deriving from the agile manifesto [7], forming a 5x5 matrix. In most of the cells of this matrix there have been assigned agile practices. The assesement of agility takes place at each level by measuring the practices adopted by a team. Before moving to the next level the team needs to fulfill all the practices of the current one.

#### Agile Levels

- Level 1 - Collaborative
- Level 2 - Evolutionary
- Level 3 - Effective
- Level 4 - Adaptive
- Level 5 - Ambient

#### Agile Principles

- Embrace Change to Deliver Customer Value
- Plan and Deliver Software Frequently
- Human Centric
- Technical Excellence
- Customer Collaboration

### 2.3.11 OPS Framework

Soundararajan [34] created the Objectives, Principles and Strategies (OPS) Framework in order to assess the “goodness” of an agile methodology. It is an evolution of the work done by Arthur and Nance [4] and Sidky [31]. The focus of this tool is mainly on eXtreme Programming, Feature Driven Development, Lean, Crystal and any tailored instances of them.

In order to achieve this the framework examines the methodology based on 3 aspects:

- Adequacy - Sufficiency of the method with respect to meeting its stated objectives.
- Capability - Ability of an organization to provide an environment supporting the implementation of its adopted method. Such ability is reflected in the characteristics of an organization’s people, process and project.
- Effectiveness - Producing the intended or expected results. The existence of necessary process artifacts and product characteristics indicate levels of effectiveness.

The framework identifies a) objectives of the agile philosophy b) principles that support the objectives c) strategies that implement the principles d) linkages that relate objectives to principles, and principles to strategies e) indicators for assessing the extent to which an organization supports the implementation and effectiveness of those strategies The OPS Framework identifies

- Objectives of the agile philosophy - “something aimed at or striven for” as defined by Arthur and Nance [4]
- Principles - what rules a process in order to achieve an objective according to Arthur and Nance [4]
- Strategies - the implementations of the principles (i.e. they are the means for achieving the principles)
- Linkages - the connectors between a) the objectives and principles, b) the principles and the strategies. The linkages show the path in order to assess the adequacy, capability and effectiveness of the method used.
- Indicators for assessing the extent to which an organization supports the implementation and effectiveness of those strategies - In order to measure the capability and the effectiveness the strategies use properties which contain a number of questions. These properties differ for the capability and the effectiveness. Indicator is named the combination of a strategy with a property. They are directly measurable and are tailored to assess the strategies

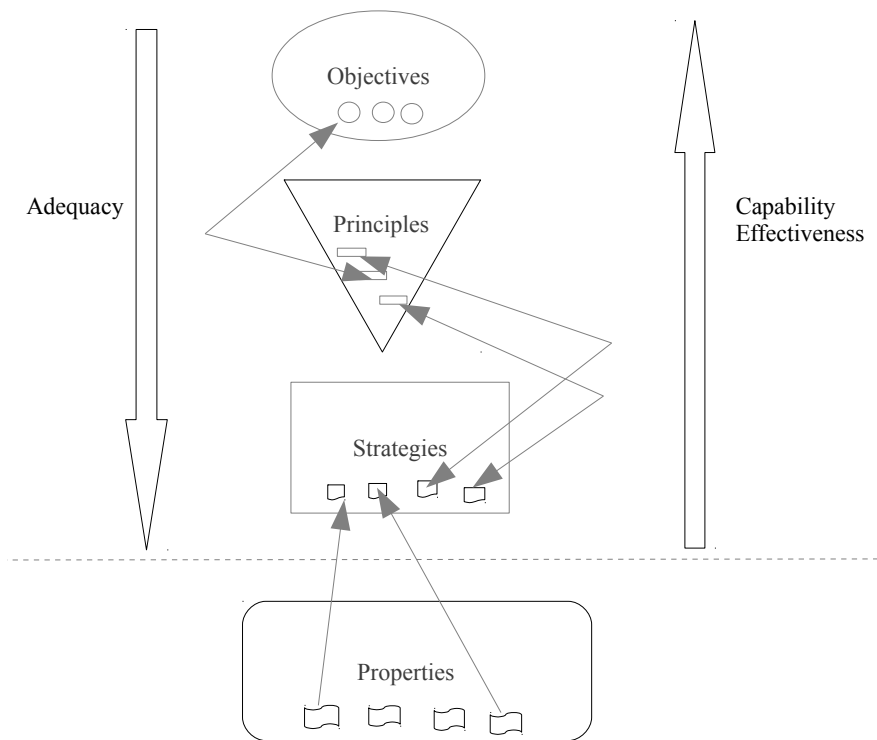
In total 5 objectives, 9 principles, 17 strategies 54 linkages and 80 indicators are identified. Also check Figure 2.5.

Individuals and Interactions	Human Centric	Frequent Delivery of Working Software	Iterative Progression
		Technical Excellence	Incremental Development
	Value-driven		Short Delivery Cycles
		Simplicity	Evolutionary Requirements
Working Software	Minimal Waste	Empowering Teams of Motivated Individuals	Continuous Feedback
		Constant Development Pace	Refactoring
	Maximal Adaptability	Accommodating Change	Test First Development
Customer Collaboration		Continual Stakeholder Communication and Collaboration	Self-Managing Teams
			Continuous Integration
			Constant Velocity
			Minimal Documentation
			High-bandwidth Communication
			Retrospection
	Continuous Innovation And Learning	Frequent Reflection and Improvement	Client-driven iterations
Responding to Change		Striving For Customer Satisfaction	Appropriate distribution of expertise
			Configuration Management
			Adherence to Standards
<b>Agile Values</b>	<b>Objectives</b>	<b>Principles</b>	<b>Strategies</b>

**Figure 2.5:** Objectives, Principles, and Strategies identified by the OPS Framework

The OPS Framework works on a top down traversal for assessing the adequacy. One should identify the objectives followed by the method and based on the linkages move to the principles and then to strategies. On the other hand for assessing the capability and effectiveness the tool works on a bottom up traversal. Based on the strategies identified during the adequacy analysis one should answer the a set of questions for the capability and to another set of questions for the effectiveness. Then the measurements are propagated to the upper levels through the linkages. For a better understanding one can see Figure 2.6.



**Figure 2.6:** OPS Framework

# 3

## Case Study

### 3.1 Problem

#### 3.1.1 Research Questions

1. In what ways do the tools correlate?
2. How much complete are the tools in measuring agility?
3. Can the tools be combined in a way so that they can produce a more complete approach on measuring agility?

### 3.2 Method

#### 3.2.1 Case Study Design

#### 3.2.2 Case and Subjects Selection

#### 3.2.3 Observations

Company F<sup>1</sup> was willing to participate in this case study in order to measure how agile are its teams. With previous work experience in this company I had an insight of how the teams work and what each of them wants to achieve each one of them. Nevertheless I spent 3 months working closely with the teams almost daily for the needs of the master thesis.

---

<sup>1</sup>F is the first letter of the company's name

### 3.2.4 Company F

Company F is a United States company which acts in the POS <sup>2</sup> area. With the development of some new products the company had a 300% increase in the size of the development and QA departments resulting in the need for organizing better the development and release processes. In addition the increasing requests of new features in the company's systems requires a more efficient way in delivering them to the customers and also maintaining the quality of the products.

### 3.2.5 Methodology F

In general, company F does not follow a specific agile methodology, but rather a tailored mix of the most famous one which suits the needs of each team. Methodology F, as we can name it, embraces the practices displayed in Table 3.1 from the various agile methodologies, some of the them in bigger and some of them in a smaller extent. For identifying these methodologies the analysis made by [18] was used. The results were verified by the agile coach.

Method	Practice
<b>XP</b>	a) Small Releases b) Simple design c) Refactoring d) Collective ownership e) Continuous integration f) 40-hour week g) Coding standards
<b>FDD</b>	a) Developing by feature b) Feature teams c) Regular build schedule d) Inspections e) Configuration management
<b>Lean</b>	a) Empower the team b) Build Integrity In c) Amplify learning d) Eliminate waste

**Table 3.1:** Practices embraced in methodology F

### 3.2.6 Teams

There are four development teams, each for a product of the company. Some of the teams have mixed members of developers and testers. In the Tables 3.2, 3.3, 3.4, 3.5, one can see the structure of the teams.

---

<sup>2</sup>Point Of Sales

<b>Team Size</b>	7
<b>Roles</b>	Team Leader (1) Developers (4) Testers (3)
<b>Development Process</b>	Method A
<b>Area</b>	Mobile
<b>Tools used</b>	Perforce Titanium
<b>Iteration length</b>	2-3 weeks

Table 3.2: Team A - Profile

<b>Team Size</b>	8
<b>Roles</b>	Team Leader (1) Developers (5) Testers (2)
<b>Development Process</b>	Method B
<b>Area</b>	Java
<b>Tools used</b>	Perforce Eclipse IDE
<b>Iteration length</b>	2-3 weeks

Table 3.3: Team B - Profile

<b>Team Size</b>	3
<b>Roles</b>	Team Leader (1) Developers (1) Testers (1)
<b>Development Process</b>	Method C
<b>Area</b>	Java
<b>Tools used</b>	Perforce Eclipse IDE
<b>Iteration length</b>	4-5 weeks

Table 3.4: Team C - Profile

<b>Team Size</b>	17
<b>Roles</b>	Team Leader (1) Developers (9) Testers (7)
<b>Development Process</b>	Method D
<b>Area</b>	Java
<b>Tools used</b>	Perforce Eclipse IDE
<b>Iteration length</b>	2-4 weeks

Table 3.5: Team D - Profile

### 3.2.7 Products

Company F has developed a few products which belong in the following four areas a) desktop b) mobile c) cloud d) platforms. The names given are respective to the name of the teams that develop them.

- Product A - A series of three mobile applications which offer services to stores or customers of stores.
- Product B - It is a cloud application which offers services to product A and product D. The web functionality of product C is limited.

- Product C - It is a platform used only by the company's employees. It supports services though which are necessary for product D.
- Product D - It is the main product of the company which is mostly used. The rest of the products were developed in order to support it and expand its functionalities.

### 3.3 OPS

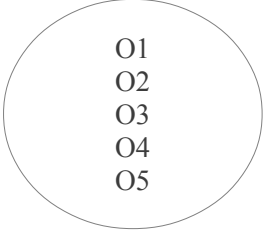
#### 3.3.1 Introduction

In order to measure the adequacy, the capability and the effectiveness of methodology F, the described method by Soundararajan [34] was followed.

#### 3.3.2 Adequacy Assessment

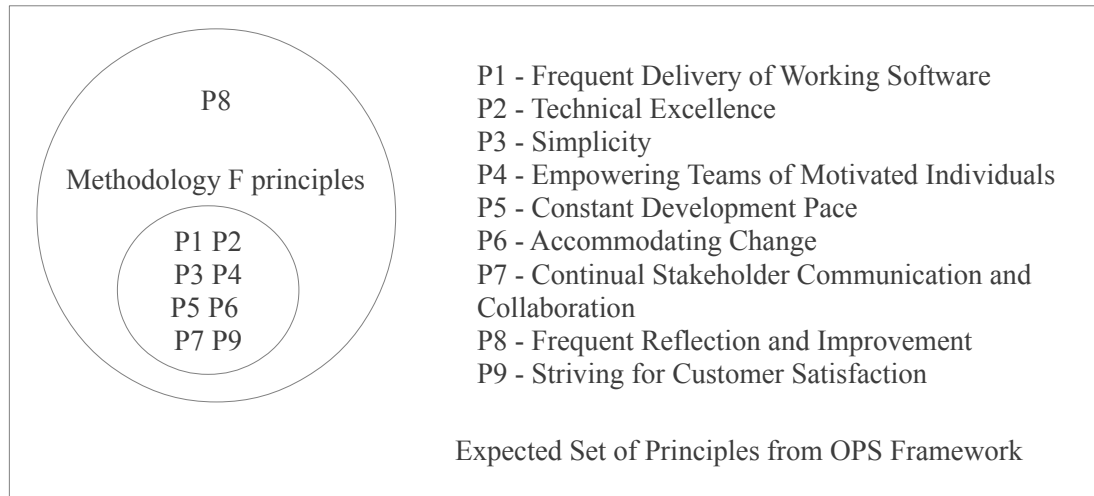
In order to assess the adequacy of methodology F a top-down traversal was used as it can be seen in Figure 2.6. For the analysis of each objective, principle and strategy the analysis of agile methodologies was followed based on Koch [18].

Initially the objectives fulfilled by methodology F were identified. As one can see in Figure 3.1 all five objectives instructed by the OPS Framework are followed.

Method F Objectives	Objectives from OPS Framework
	
O1	O1 – Human-centric
O2	O2 – Value-driven
O3	O3 – Minimal Waste
O4	O4 – Maximal Adaptability
O5	O5 – Continuous Innovation and Learning

**Figure 3.1:** Objectives identified in methodology F

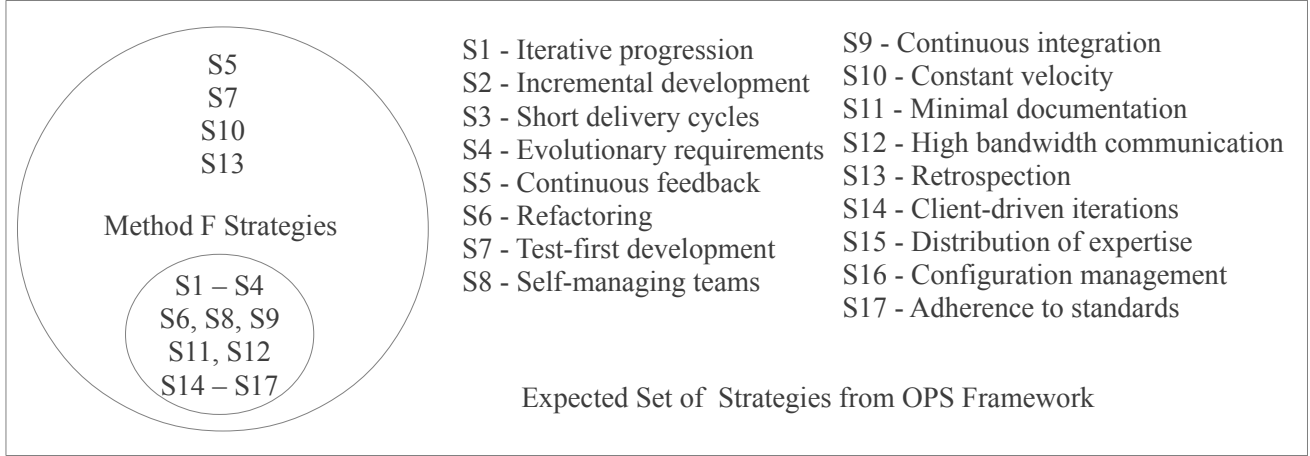
Based on the objectives and following the linkages from them, the principles were identified. As one can see in Figure 3.1 methodology F does not follow the “Frequent Reflection and Improvement” principle because the organization rarely does it re-examine the development process in order to improve it. It is worth mentioning that the “Empowering teams of Motivated Individuals” principle is not entirely followed either, but it differs among the teams. Every team is built with motivated individuals, some to a more and some to a lesser extent.



**Figure 3.2:** Principles identified in methodology F

Following the linkages from the principles the strategies for implementing them were identified. As it can be seen in Figure 3.3 methodology F does not support

- **Continuous feedback** - The organization does not have a defined process for getting a feedback from the customers of the company. From time to time the managers of the various departments of the company have personal conversations with the customers. If any issue arises, then they inform the development and QA departments in order to identify the problem and fix it.
- **Test-first development** - None of the team members writes tests before starting coding.
- **Constant velocity** - The organization does not measure the velocity of the teams. In general the pace of development and integration and deployment is based on the needs of the customers and the capability of the developer. No one has to finish a specific amount of work in each iteration. What is only wanted, is the functionality to be delivered when it is scheduled.
- **Retrospection** - Although there is a tendency to change this, the teams do not have a process for retrospection. The team members unconsciously consider that they are doing fine, unless a team leader or the manager of the department tells them the opposite.



**Figure 3.3:** Strategies identified in methodology F

Reflecting on the above analysis, we see that methodology F lacks four strategies which are important for its improvement. Without retrospection the team members do not always identify and discuss their mistakes, while without a well defined process for continuous feedback any issues that might arise may go unnoticed for quite some time. In total, methodology F is missing four strategies out of seventeen which is a lot. As a result the adequacy of methodology F is questionable.

### 3.3.3 Capability and Effectiveness Assessment

In order to assess the capability and effectiveness of methodology F a bottom-up traversal was used as it can be seen in Figure 2.6. Based on the adequacy analysis performed in subsection 3.3.2 the assessment of the capability and effectiveness started from the strategies and following the linkages moved to the principles and then to the objectives. As an observer of the teams I filled in the questions from the capability and effectiveness hierarchy. The scales used were the ones proposed by Soundararajan [34]. A ternary scale for the capability and a scale between 1-5 for the effectiveness (see Table 3.6 and Table 3.7 respectively). For the interpretation of the scores for both the capability and effectiveness the scale in Table 3.8 was used. All the questions answered can be seen in Capability Hierarchy and Effectiveness Hierarchy Appendices.

Answer	Score	Answer	Score	Score Range	Interpretation
Yes/Fully	5	Maximally	5	4.5 - 5.0	Maximal
Partiallty	3	Considerably	5	3.5 - 4.4	Considerable
No/Marginally	1	Moderately	3	2.5 - 3.4	Moderate
		Somewhat	2	1.5 - 2.4	Somewhat
		Marginally	1	1 - 1.4	Marginal

**Table 3.6:** Scale for Capability**Table 3.7:** Scale for Effectiveness**Table 3.8:** Scale for interpreting final scores

### General remarks about the teams

#### Team A Assessement

<b>Capability:</b> 2.3 - Somewhat	<b>Effectiveness:</b> 2.3 - Somewhat
-----------------------------------	--------------------------------------

Team A is a team with seven members. One team leader, four developers and three testers. The product is a series of mobile applications either for the stores, or the customers of the stores.

#### Capability Analysis

##### Strategies

The team differs from the rest in the *adherence to standards* strategy. This is because there are no specified techniques yet to identify the features for the applications. Since they are fairly new the team members are still in the process of working it out. In addition some of the team members do not follow the coding standards of the company resulting in badly written code which has been refactored already twice because of this. As far as the *evolutionary requirements* strategy is concerned, as it was mentioned above, the applications are failry new and the team is still working on improving the process of identifying and modifying the requirements.

##### Principles

The team is in accordance with the rest.

##### Objectives

The team is in accordance with the rest.

#### Effectiveness Analysis



**Strategies**

The *high bandwidth communication* has the lowest value, mostly because the developers work on different applications and rarely have to communicate with each other, unless they need some help.

**Principles**

The team is in accordance with the rest.

**Objectives**

The team is in accordance with the rest.

**Team B Assessment**

<b>Capability:</b> 2.6 - Moderate	<b>Effectiveness:</b> 2.4 - Somewhat
-----------------------------------	--------------------------------------

Team B is a team with an average size. It consists of eight members. One team leader, five developers and two testers. The product provides services which are used by product A and product C. The application itself is in Amazon Web Services (AWS).

**Capability Analysis****Strategies**

The team differs from the rest in the *self managing teams* strategy since this team always consisted of the most motivated but also valuable technically developers. Since the decisions for the new features are taken 85% from the team itself, the client driven iterations and incremental development are high.

**Principles**

The team has the highest value for the empowering teams of motivated individuals principle for the reason that was mentioned right above. In addition since the customer are the members themselves the striving for customer satisfaction principle has the highest value.

**Objectives**

The team has the highest values in all the objectives. This is because.....

**Effectiveness Analysis****Strategies**

The team differs in the client driven iterations strategy for the reasons mentioned above.

**Principles**

The team differs in theempowering teams of motivated individuals principle for the reasons mentioned above.

**Objectives**

The team is in accordance with the rest.

**Team C Assesement**

<b>Capability:</b> 2.3 - Somewhat	<b>Effectiveness:</b> 2.5 - Moderate
-----------------------------------	--------------------------------------

Team C is the smallest team of the company, with only three team members. One team leader, one developer and one tester. The product developed is only internally used by the company iteself, nevertheless, it is of great importance since without it the non developer and tester employees would not be able to work. In addition, it is needed by the rest products to properly work.

**Capability Analysis****Strategies**

The team differs from the rest in the *client driven iterations*, the *short delivery cycles* and the *minimal documentation* strategies. This is the only team with such a high score for the client driven iteration. Since the product is only used by the employees of the company then the identification and prioritization of the features is done constantly and immediately. As far as the short delivery cycles, since there are no “external” customers waiting for it, the releases cycle depends only on the features being worked. If nothing important is under development, there is no rush to release it. Finally this is the product with the least documentation maintained, for the same reasons that the other two aformentioned strategies differ.

**Principles**

The team is in accordance with the rest.

**Objectives**

The team is in accordance with the rest.

**Effectiveness Analysis****Strategies**

The team differs follows the same trend with team D for the continuous integration. This is because there are unit test suites and no automated build environment. As far as the high-bandwidth communication is concerned, the team has the highest score, due to the fact the the size of the team is minimal and all the communication is fast and easy.

The customer satisfaction is high as well, since “the product is built right”, reinforcing not only this strategy, but also the client-driven iteration one.

### Principles

As it was mentioned before, due to the team size the individuals are extremely motivated and feel responsible for the product.

### Objectives

The team is in accordance with the rest.

### Team D Assessment

<b>Capability:</b> 2.4 - Somewhat	<b>Effectiveness:</b> 2.2 - Somewhat
-----------------------------------	--------------------------------------

Team D is the biggest team of the company. It consists of 17 members. One team leader, nine developers and seven testers. The product created is the main product of the company and thus it has many needs for people working on it.

### Capability Analysis

#### Strategies

The team is in accordance with the rest.

#### Principles

The team is in accordance with the rest.

#### Objectives

The team is in accordance with the rest.

### Effectiveness Analysis

#### Strategies

This differs from the rest in the *distribution of expertise*, since it is the biggest team the average of the team member’s capabilities is low and as result the score for this strategy is the lowest as well.

#### Principles

The team is in accordance with the rest.

### Objectives

The team is in accordance with the rest.

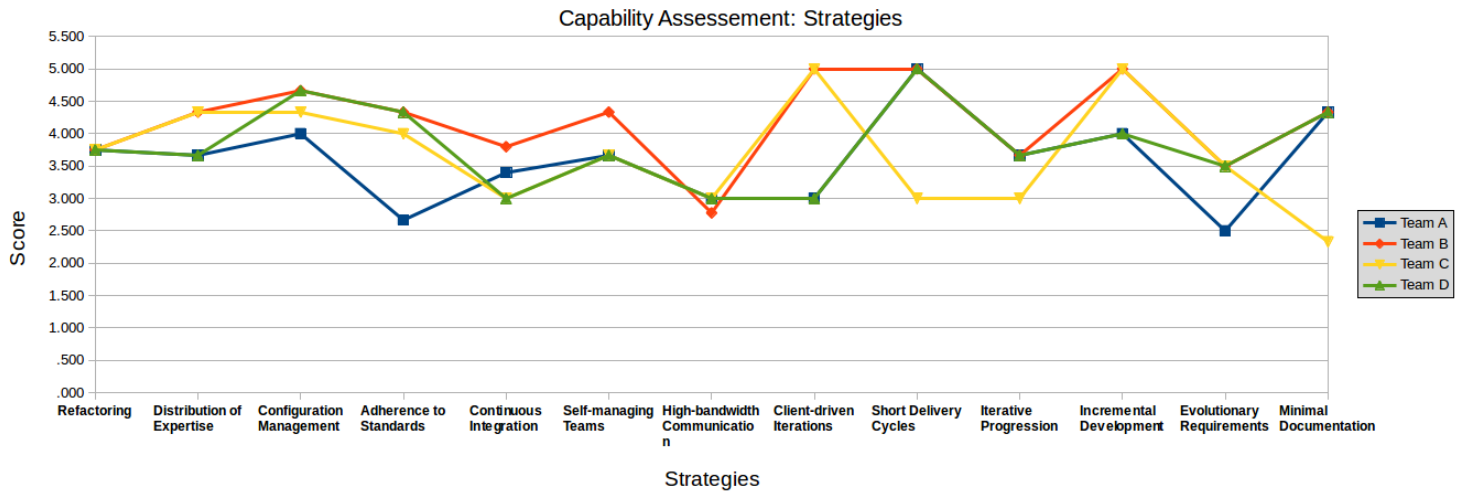


Figure 3.4: Capability - Strategies level

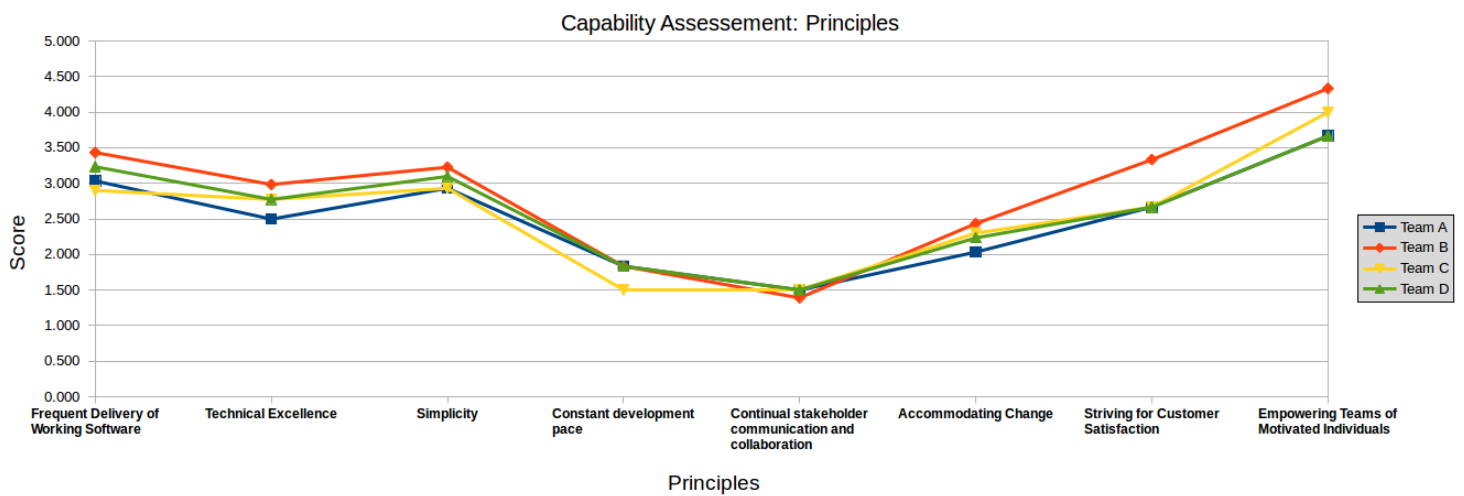


Figure 3.5: Capability - Principles level

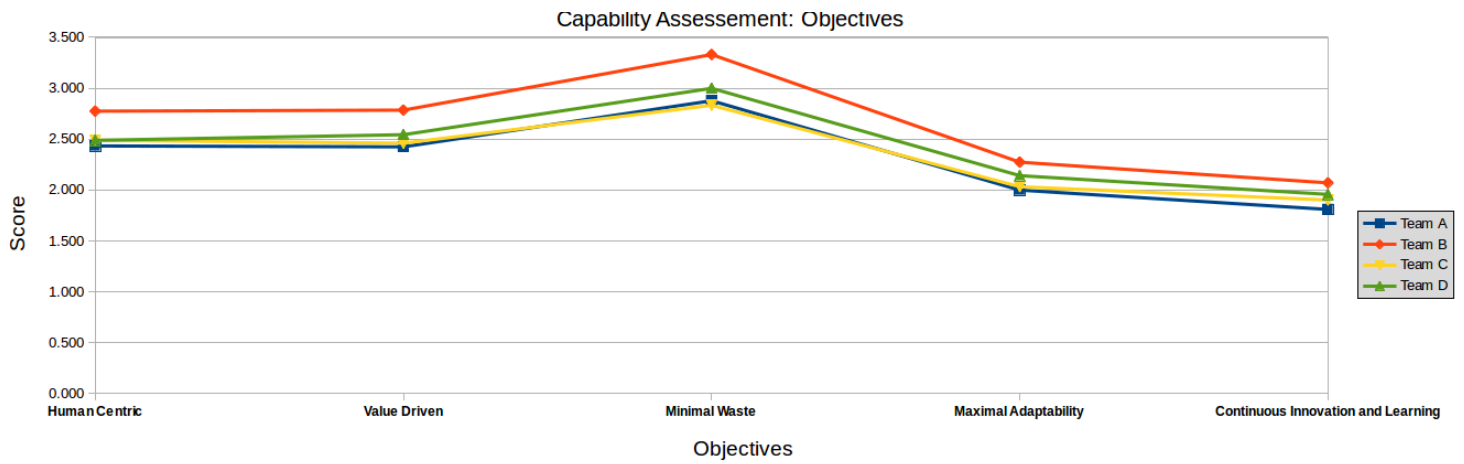


Figure 3.6: Capability - Objectives level

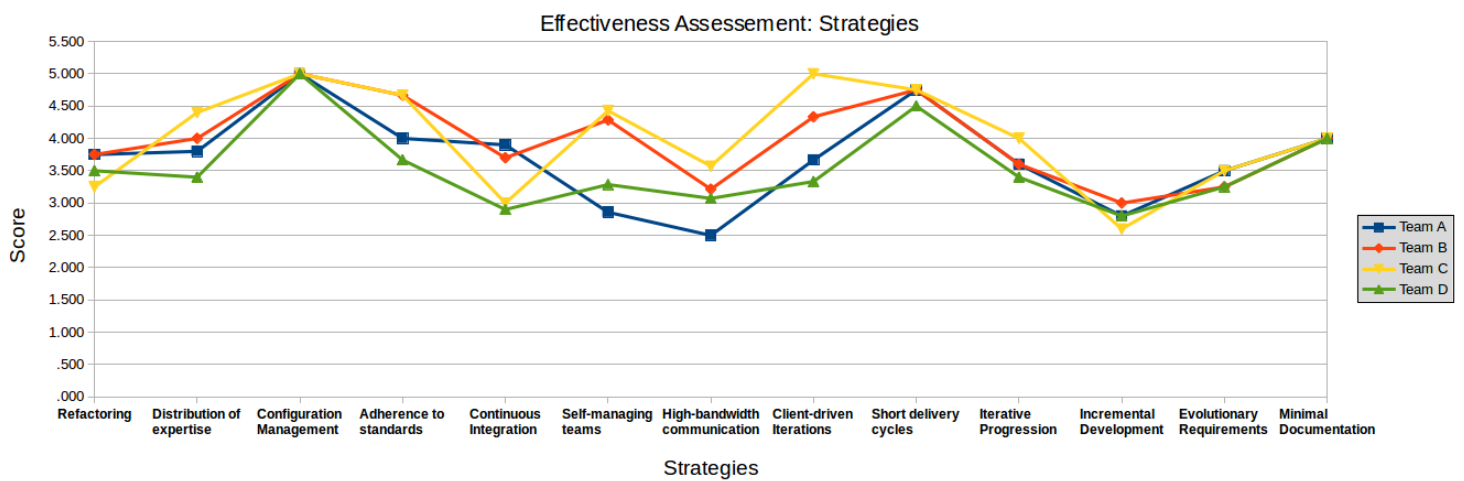


Figure 3.7: Effectiveness - Strategies level

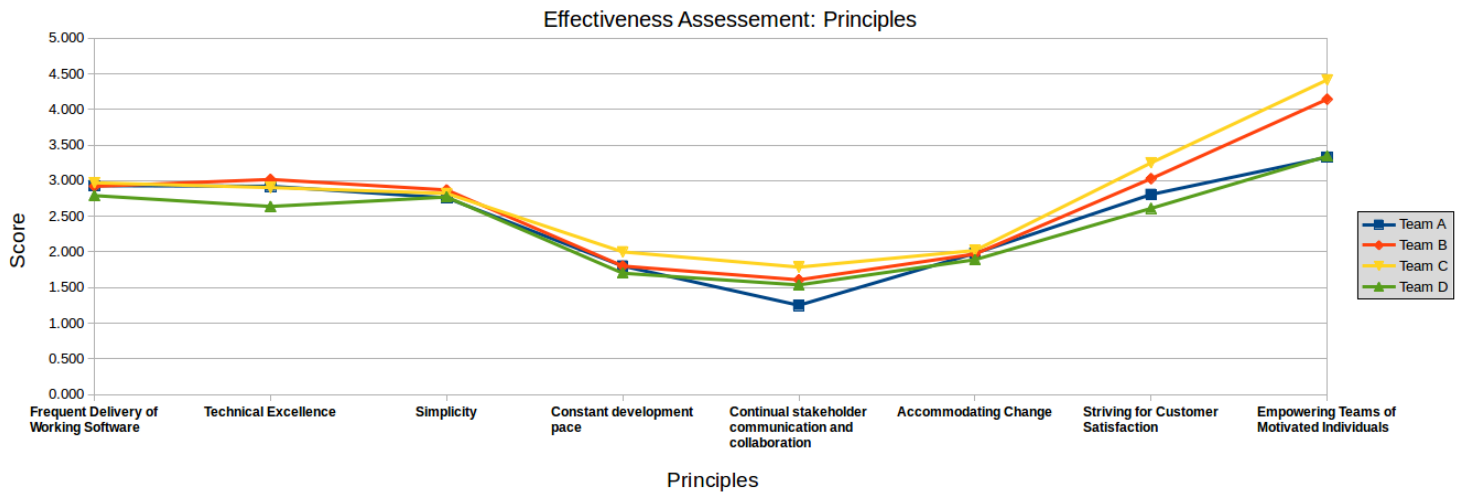


Figure 3.8: Effectiveness - Principles level

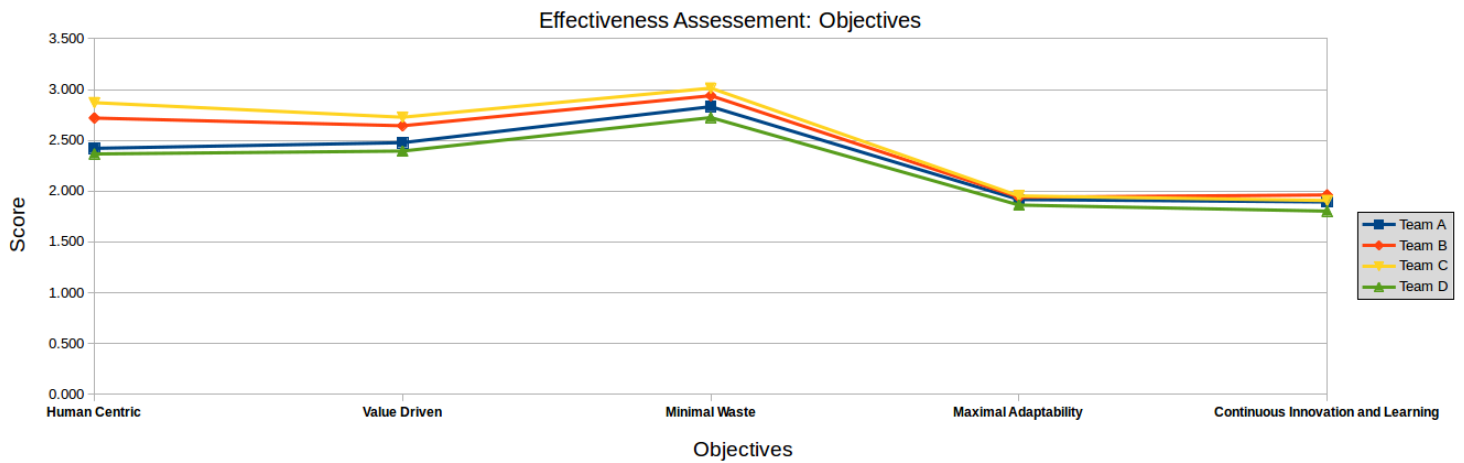


Figure 3.9: Effectiveness - Objectives level

### 3.3.4 Results Verification

## 3.4 Perceptive Agile Measurement

For assessing the agility by using the Perceptive Agile Measurement (PAM) tool, the teams were asked to answer on the survey seen in the Perceptive Agile Measurement Appendix. The answers were on a Likert scale 1-7.

**Team A Assesement**

<b>Score:</b> value
---------------------

**Team B Assesement**

<b>Score:</b> value
---------------------

**Team C Assesement**

<b>Score:</b> value
---------------------

**Team D Assesement**

<b>Score:</b> value
---------------------

**3.4.1 Results Verification****3.5 Team Agility Assessment****3.5.1 Results Verification**

Team Agility Assessment

**3.6 Thoughtworks****3.6.1 Results Verification****3.6.2 OPS**

Figure 2.5

**3.6.3 PAM****3.6.4 Leffingwell****3.7 Discussion**

# 4

## Tools Completeness

**RQ #2 - How much complete are the tools in measuring agility?**

### 4.1 Team Agility Assessment Areas

Team Agility Assessment (TAA) does not state covering specific agile practices, but rather areas important for a team. It focuses on product ownership for Scrum teams but also on the release and iteration planning and tracking. The team factor plays a great role but also the development practices and the working environment. Automated testing is important here as well. Finally it is worth mentioning that it is the only tool focusing so much on the release planning. In Table 4.1 one can see TAA's areas.

TAA Areas
<ul style="list-style-type: none"><li>• Product Ownership</li><li>• Release Planning and Tracking</li><li>• Iteration Planning and Tracking</li><li>• Team</li><li>• Testing Practices</li><li>• Development Practices / Infrastructure</li></ul>

**Table 4.1:** Areas covered by TAA



## 4.2 Perceptive Agile Measurement Practices

The Perceptive Agile Measurement (PAM) tool focuses on the iterations during software development but also on the stand-up meetings among the team members, their collocation and the retrospectives they have. The customers access and their acceptance criteria have a high significance as well. Finally the continuous integration and the automated unit testing are considered crucial in order to be agile. In Table 4.2 one can see PAM's practices.

PAM Practices
<ul style="list-style-type: none"> <li>• Iteration Planning</li> <li>• Iterative Development</li> <li>• Continuous Integration and Testing</li> <li>• Co-Location</li> <li>• Stand-up Meetings</li> <li>• Customer Access</li> <li>• Customer Acceptance Tests</li> <li>• Retrospectives</li> </ul>

**Table 4.2:** Agile practices covered by PAM

## 4.3 Objectives, Principles, Strategies Practices

Objectives, Principles, Strategies (OPS) Framework is the successor of the Objectives, Principles, Practices (OPP) Framework [35]. OPP identified 27 practices as implementations of the principles which later on were transformed into 17 strategies. In Table 4.3 one can see OPP's practices.

OPP Practices	
<ul style="list-style-type: none"> <li>• Iterative and Incremental Development</li> <li>• Continuous Feedback</li> <li>• Evolutionary Requirements</li> <li>• Smaller and Frequent Product Releases</li> <li>• Customer/User Acceptance Testing</li> <li>• Frequent Face-to-Face Communication</li> <li>• Refactoring</li> <li>• Automated Test Builds</li> <li>• Software Configuration Management</li> <li>• Test Driven Development</li> <li>• Iteration Progress Tracking and Reporting</li> <li>• Code Ownership</li> <li>• Retrospectives Meetings</li> <li>• Just-in-Time Refinement of Features /Stories/Tasks</li> </ul>	<ul style="list-style-type: none"> <li>• Appropriate Distribution of Expertise</li> <li>• Self-Organizing Teams</li> <li>• Client-Driven Iterations</li> <li>• Product Backlog</li> <li>• Agile Project Estimation</li> <li>• Adherence to Coding Standards</li> <li>• Physical Setup Reflecting Agile Philosophy</li> <li>• Daily Progress Tracking Meetings</li> <li>• Minimal or Just Enough Documentation</li> <li>• Minimal Big Requirements Up Front and Big Design Up Front</li> <li>• Collocated Customers</li> <li>• Constant Velocity</li> <li>• Pair Programming</li> </ul>

**Table 4.3:** Agile practices covered by OPP

## 4.4 Practices Covered Between The Tools

As it can be clearly seen between Tables 4.3, 4.2 4.1 OPP and as a consequence OPS covers more agile practices than the other tools.

In the next pages follows a mapping between OPP and PAM (see Table 4.4) and OPP and TAA (see Table 4.5).

Some of the OPP practices though have abstracted to OPS strategies in order to avoid repetition of the questions' mapping and in order to better reflect the OPS Framework. The OPP practices a) *Frequent Face-to-Face Communication* b) *Physical*

#### 4.4. PRACTICES COVERED BETWEEN CHAPTER 4.5 TOOLS COMPLETENESS

*Setup Reflecting Agile Philosophy* c) *Collocated Customers* have been abstracted to the OPS strategy *High-Bandwidth Communication* [34, p. 57]. In the same way the OPP *Automated test builds* practice has been abstracted to the OPS strategy *Continuous Integration* [34, p. 57].

The connection between the practices and strategies is done based on the questions of each tool. The aforementioned connections are depicted with colours. When a practice has more than one colour, it is because it covers more practices from the other tool (The colours and symbols among Tables 4.4, 4.5 are randomly selected and do not imply any connection between the two tables).

OPP/OPS	PAM
Iteration Progress Tracking and Reporting ✦	Iteration Planning ✦
Iterative and Incremental Development ✦ ✦	Iterative Development ✦
Continuous Integration ✦	Continuous Integration and Testing ✦
Software Configuration Management ✦	Co-Location ✦
Test Driven Development ✦ ✧	Stand-up Meetings ✧
High-Bandwidth Communication ✦ ✦	Customer Access ✦
Daily Progress Tracking Meetings ✧	Customer Acceptance Tests ✧
Client-Driven Iterations ✦ ✦	Retrospectives ✦
Continuous Feedback ✦	
Customer/User Acceptance Testing ✧	
Retrospectives Meetings ✦	
Self-Organizing Teams ✦	

**Table 4.4:** Relation of OPP/OPS and PAM practices

OPP/OPS	TAA
<p>Iterative and Incremental Development *</p> <p>Product Backlog *</p> <p>Smaller and Frequent Product Releases *</p> <p>Customer/User Acceptance Testing ♦ *</p> <p>Constant Velocity ♦</p> <p>Iteration Progress Tracking and Reporting ♦</p> <p>Self- Organizing Teams ☒ * ♦ ✚</p> <p>Appropriate Distribution of Expertise ☒</p> <p>High-Bandwidth Communication ☒</p> <p>Daily Progress Tracking Meetings ☒</p> <p>Retrospectives Meetings ☒ ♦ *</p> <p>Test Driven Development ✻</p> <p>Refactoring ✚</p> <p>Software Configuration Management ✚</p> <p>Adherence to Coding Standards ✚</p> <p>Pair Programming ✚</p> <p>Continuous Integration ✚ ✻</p>	<p>Product Ownership *</p> <p>Release Planning and Tracking *</p> <p>Iteration Planning and Tracking ♦</p> <p>Team ☒</p> <p>Testing Practices ✻</p> <p>Development Practices/Infrastructure ✚</p>

Table 4.5: Relation of OPP/OPS and TAA practices/areas

## 4.5 Mapping of questions from the PAM and TAA tools

PAM has divided its questions based on agile practices, while on the other hand TAA has divided them based on areas considered important. In the next pages there is a mapping of the questions used from the PAM and TAA tools, with the practices from OPP and strategies from OPS. As one can see from the tables above, while all practices/areas from PAM and TAA are mapped to OPP and OPS, not all of their questions are under OPP practices or OPS strategies. This can be explained due to the different perception/angle, of the creators of the tools, of what is important for an organization to be agile.

## Introduction

The questions among the tools will be match based on whether they are covered directly, relevantly, or not at all. Direct match will be considered the one where a question from a tool is almost similar with one from OPS. Relevant match will be considered the one where a question of a tool does not exist in OPS, but its practice does exist in OPS. Non relevant match will be considered the one where a question cannot be matched at all in OPS.

In order to distinguish the questions among the tools the following annotation has been used.

- ★ Team Agility Assessment - Direct match
- ☆ Team Agility Assessment - Relevant match
- ✱ Team Agility Assessment - Non Relevant
- ✱ Perceptive Agile Management - Direct match
- ✱ Perceptive Agile Management - Relevant match

### 4.5.1 Questions to Practices/Strategies

#### Iterative and Incremental Development

- ★ Stories sufficiently elaborated prior to planning meetings
- ✱ The software delivered at iteration end always met quality requirements of production code
- ✱ We implemented our code in short iterations
- ✱ The team rather reduced the scope than delayed the deadline
- ✱ We kept the iteration deadlines
- ✱ At the end of an iteration, we delivered a potentially shippable product
- ✱ Working software was the primary measure for project progress

#### Product Backlog

- ★ Backlog prioritized and ranked by business value
- ★ Backlog estimated at gross level

#### Smaller and Frequent Product Releases

- ★ The team has small and frequent releases



- ✧ The effort estimates for the iteration scope items were modified only by the technical team members

#### **Self-Organizing Teams**

- ★ Team self-polices and reinforces use of agile practices and rules
- ★ Team leads communication; communication not managed
- ★ Team defines, estimates, and selects their own work (stories and tasks)
- ★ Team develops and manages iteration backlog
- ★ Team manages interdependencies and constraints
- ★ Team has administrative access to their own workstations
- ★ Team has administrative control over their development environment
- ☆ Team is 100% dedicated to the release (no time-slicing)
- ☆ Team members complete commitments
- ☆ The team has an effective channel for obstacle escalation
- ☆ The team has a common language and metaphor to describe the release
- ✱ Each developer signed up for tasks on a completely voluntary basis

#### **Appropriate Distribution of Expertise**

- ★ Team is cross-functional with integrated product owner, development, documentation and QA
- ★ Team manages interdependencies and constraints

#### **High-Bandwidth Communication**

- ★ Team is collocated
- ★ Team works in a physical environment that fosters collaboration
- ★ Team Coach/Scrum Master exists, is full-time, and is effective
- ✱ The customer was reachable
- ✱ The developers could contact the customer directly or through a customer contact person without any bureaucratic hurdles
- ✱ The developers had responses from the customer in a timely manner
- ✱ Developers were located majorly in ...

- \* All members of the technical team (including QA engineers, db admins) were located in ...
- \* Requirements engineers were located with developers in ...
- \* The project/release manager worked with the developers in ...
- \* The customer was located with the developers in ...

#### **Daily Progress Tracking Meetings**

- ★ Daily standup on time, fully attended and effectively communicates
- \* Stand up meetings were extremely short (max. 15 minutes)
- \* Stand up meetings were to the point, focusing only on what had been done and needed to be done on that day
- ✧ All relevant technical issues or organizational impediments came up in the stand up meetings
- ✧ Stand up meetings provided the quickest way to notify other team members about problems
- ✧ When people reported problems in the stand up meetings, team members offered to help instantly

#### **Retrospective Meetings**

- ★ Team inspects and adapts (continuous improvement) the overall process
- ★ Team inspects and adapts (continuous improvement) the Iteration Plan
- ★ Team inspects and adapts (continuous improvement) the release plan
- ★ Iteration review meeting attended and effective
- ★ Release review meeting attended and effective
- \* How often did you apply retrospectives?
- \* The retrospectives helped us become aware of what we did well in the past iteration(s)
- \* The retrospectives helped us become aware of what we should improve in the upcoming iteration(s)
- \* All team members actively participated in gathering lessons learned in the retrospectives



- ✧ In the retrospectives (or shortly afterwards), we systematically assigned all important points for improvement to responsible individuals
- ✧ Our team followed up intensively on the progress of each improvement point elaborated in a retrospective

#### **Test Driven Development**

- ★ Unit tests written before development
- ★ Acceptance tests written before development
- ★ 100% automated unit test coverage
- ✧ New code was written with unit tests covering its main functionality
- ✧ Automated unit tests sufficiently covered all critical parts of the production code
- ✧ The customer provided a comprehensive set of test criteria for customer acceptance
- ✧ The implemented code was written to pass the test case
- ✧ There were enough unit tests and automated system tests to allow developers to safely change any code

#### **Refactoring**

- ★ Refactoring is continuous
- ★ Team is permitted to refactor anywhere in the code base

#### **Software Configuration Management**

- ★ Source control system exists

#### **Adherence to Coding Standards**

- ★ Coding standards exist and applied
- ☆ Adequate and effective code review practices

#### **Pair Programming**

- ★ Pair programming is practised

#### **Continuous Integration**

- ★ Developers integrate code multiple times per day
- ★ Stories accepted and demonstrated on integrated build
- ★ Continuous build with 100% successful builds

- ★ Automated acceptance tests
- ★ Identical builds for developers' workstations
- \* The team integrated continuously
- \* Developers had the most recent version of code available
- \* Code was checked in quickly to avoid code synchronization/integration hassles
- \* All unit tests were run and passed when a task was finished and before checking in and integrating
- ✧ For detecting bugs, test reports from automated unit tests were systematically used to capture the bugs

#### **Continuous Feedback**

- \* The feedback from the customer was clear and clarified his requirements or open issues to the developers

#### **Client-driven Iterations**

- \* The customer picked the priority of the requirements in the iteration plan
- \* When the scope could not be implemented due to constraints, the team held active discussions on re-prioritization with the customer on what to finish within the iteration

#### **4.5.2 Non Relevant Questions**

- ✧ Product owner defines acceptance criteria for stories
- ✧ Product owner and stakeholders participate at iteration and release planning
- ✧ Product owner and stakeholders participate at iteration and release review
- ✧ Product owner collaboration with team is continuous
- ✧ Release theme established and communicated
- ✧ Release planning meeting attended and effective
- ✧ Release backlog defined
- ✧ Release backlog ranked by priority
- ✧ Release backlog estimated at plan level
- ✧ Release progress tracked by feature acceptance

- ❄ Team completes and product owner accepts the release by the release date
- ❄ Team meets its commitments to release
- ❄ Work is not added by the product owner during the iteration
- ❄ Team completes and product owner accepts the iteration
- ❄ The whole team is present at release planning meetings
- ❄ Team is smaller than 15 people
- ❄ All testing is done within the iteration and does not lag behind
- ❄ Iteration defects are fixed within that iteration

## 4.6 Results

By viewing Tables 4.4, 4.5 and Section 4.5 one can clearly distinguish that OPP and consequently OPS is more complete than the others in measuring agility, covering all the areas of the PAM and TAA tools. Furthermore as it can be seen in Table 4.6 OPS covers a high percent of questions from both tools directly and relevantly. TAA has a high percent of non relevant matches mostly due to *Product Ownership* and *Release Planning and Tracking* perspectives which are not covered in such an extent from OPS. The first one can be explained by the fact that OPS covers basis methodologies for developing software such as XP, FDD, Crystal, Lean [34, p. 44] whereas *Product Ownership* refers explicitly to Scrum which is a method for managing product development [18]. On the other hand OPS considers the release cycle as smaller iteration cycles [34, p. 13] which as a consequence makes the framework to set the focus on the iterations, rather than the releases. As a result the *Release Planning and Tracking* is only covered to a small extent.

Questions Coverage		
Match	PAM	TAA
Direct Match	33/48 (68.75%)	44/67 (65.6%)
Relevant Match	15/48 (31.25%)	5/67 (7.4%)
Non Relevant	0/48 (0%)	18/67 (27%)

**Table 4.6:** Questions Coverage from OPS

# Appendices

# A

## The Capability and Effectiveness Hierarchy for Company F

### A.1 Capability Hierarchy

- Refactoring
  - Support for Refactoring
    - \* Is refactoring an expected activity?
    - \* Is it feasible to implement code refactoring?
    - \* Is it feasible to implement architecture refactoring?
  - Buy-in for Refactoring
    - \* Are the teams receptive to implementing refactoring?
    - \* Is the management receptive to supporting refactoring efforts?
  - Minimizing Technical Debt
    - \* Is it expected that a well-defined process be adopted to minimize technical debt?
    - \* Is it expected that a well-defined process be adopted to manage technical debt?
    - \* Is minimizing technical debt a high priority activity?
- Distribution of expertise
  - Appropriate team composition
    - \* Is a scheme for appropriate team composition defined?
    - \* Are the requisite skillsets for particular projects identified upfront?
    - \* Is it expected that the right people be chosen to accomplish the tasks?

- Configuration Management
  - Tool Support for Configuration Management
    - \* Do tools for version control and management exist?
  - Support for Configuration Management
    - \* Is it expected that the code be kept up to date?
    - \* Is it expected that the tests be kept up to date?
    - \* Is it expected that the builds be kept up to date?
    - \* Is it expected that the release infrastructure be kept up to date?
    - \* Is it expected that the documentation be kept up to date?
- Adherence to standards
  - Identifying features
    - \* Is it expected that well-defined techniques be used to identify the features?
  - Estimation
    - \* Is it expected that a well-defined approach to estimating the amount of work to be done during each release cycle and iteration be used?
  - Requirements Prioritization
    - \* Is it expected that a well-defined approach to prioritizing bugs/enhancements, and tasks be used?
  - Feature Decomposition
    - \* Is it expected that a mechanism for decomposing the selected features to be developed during the current release cycle into bugs/enhancements be defined?
  - Coding standards
    - \* Is it expected that each team creates and adopts a set of coding standards?
    - \* Is it expected that practices such as pair-programming, collective code ownership be adopted or automated tools be used to ensure adherence to the set standards?
- Continuous Integration
  - Tool Support for Continuous Integration
    - \* Do automated test suites exist?
    - \* Does the requisite test environment exist?
    - \* Do appropriate configuration management systems exist?
  - Process Support for Continuous Integration

- \* Is continuous integration an expected activity?
  - \* Are the team members expected to integrate their code every few hours?
  - \* Is it expected that the builds, tests, and other release infrastructure be kept up to date?
  - \* Is it expected that automated test suites be developed?
  - \* Is it expected that the build process be automated?
- Buy-in for Continuous Integration
  - \* Are the teams receptive to implementing continuous integration?
- Story Completeness
  - \* Is it expected that the criteria for Done/Done be specified upfront?
- Self-managing teams
  - Team Empowerment
    - \* Are the team members expected to be involved in determining, planning, and managing their day-to-day activities?
  - Ownership
    - \* Are the team members expected to demonstrate individual or collective code ownership?
  - Performance Expectations
    - \* Is there a set of performance expectations that are agreed upon by the team and the management?
- High-bandwidth communication
  - On-site Customer
    - \* Are the customers available onsite to answer questions and provide continuous feedback?
    - \* In the absence of an onsite customer, do the customers provide feedback via other means?
  - Scheduling
    - \* Is it expected that time be allocated for Release Planning?
    - \* Is it expected that time be allocated for Iteration Planning?
    - \* Is it expected that time be allocated for Retrospection?
    - \* Is it expected that time be allocated for Daily Progress Tracking meetings?
  - Inter- and intra-team communication
    - \* Is it expected that team members communicate and collaborate with their colleagues?

- \* Do the teams have access to requisite tools to support inter- and intra-team communication?
- Physical environment
  - \* Is the physical environment conducive to supporting high bandwidth communication?
- Client-driven Iterations
  - Identifying and prioritizing features
    - \* Are the customers expected to be involved in identifying the features?
    - \* Are the customers expected to establish the priorities of the features?
- Short delivery cycles
  - Development time-frames
    - \* Is it expected that the product be developed over short delivery cycles? For example, a product increment should be released every 6 - 12 months and iterations last for four weeks or less.
- Iterative Progression
  - Planning
    - \* Is the team expected to plan for each iteration?
  - Estimation Authority
    - \* Are the developers expected to estimate the time required to complete each bug/enhancement?
  - Estimation
    - \* Is it expected that a well-defined approach to estimating the amount of work to be done during each release cycle and iteration be used?
- Incremental Development
  - Estimation Authority
    - \* Are the developers expected to estimate the time required to complete each bug/enhancement?
  - Requirements Management
    - \* Are tools available for managing the bugs/enhancements?
  - Identifying and prioritizing features
    - \* Are the customers expected to be involved in identifying the features?
    - \* Are the customers expected to establish the priorities of the features?
- Evolutionary Requirements



- Minimal Big Requirements Up Front and Big Design Up Front
  - \* Is it expected that only high level features be identified upfront?
  - \* Is it expected that an evolutionary approach to architecting the system be followed as opposed to creating the architecture upfront?
- Just-In-Time Refinement
  - \* Is it expected that the requirements be determined and refined just-in-time?
- Feature Decomposition
  - \* Is it expected that a mechanism for decomposing the selected features to be developed during the current release cycle into stories be defined?
- Minimal Documentation
  - Tool Support for Minimal Documentation
    - \* Do tools for maintaining documentation exist?
  - Process support for Minimal Documentation
    - \* Is it expected that minimal documentation be maintained?
  - Buy-in for Minimal Documentation
    - \* Are the teams receptive to maintaining minimal or just-enough documentation?

## A.2 Effectiveness Hierarchy

- Refactoring
  - Minimizing Technical Debt
    - \* To what extent do the teams manage technical debt?
    - \* To what extent do the teams minimize technical debt when developing new systems?
    - \* To what extent does the system and the development environment allow Technical Debt to be minimized?
  - Buy-in for Refactoring
    - \* To what extent does the management support the implementation of refactoring?
    - \* To what extent do the teams implement refactoring?
- Distribution of expertise
  - Process Outcomes for Distribution of Expertise
    - \* To what extent do the team members have the requisite expertise to complete the tasks assigned to them?

- \* To what extent is the work assigned to the team members commensurate with their expertise?
  - \* To what extent does the team effectively complete the work that they have committed to?
  - \* To what extent do the teams have members in leadership positions that can guide the others?
  - \* To what extent do the teams not rely on knowledge external to their teams?
- Configuration Management
  - Project Environment for Configuration Management
    - \* To what extent do teams use appropriate tools for version control and management?
- Adherence to standards
  - Estimation
    - \* To what extent are the estimates for the amount of work to be done during each iteration accurate?
  - Coding Standards
    - \* To what extent do the team members agree with the set coding standards?
    - \* To what extent do the team members adhere to the set coding standards?
- Continuous Integration
  - Project Environment for Continuous Integration
    - \* To what extent are automated test suites developed?
    - \* To what extent are the code bases not shared?
  - Bug/Enhancement Completeness
    - \* To what extent has each bug/enhancement been coded?
    - \* To what extent has each bug/enhancement been unit tested?
    - \* To what extent has each bug/enhancement been refactored?
    - \* To what extent has each bug/enhancement been checked into the code base?
    - \* To what extent has each bug/enhancement been integrated with the existing code base?
    - \* To what extent has each bug/enhancement been reviewed?
    - \* To what extent has each bug/enhancement been accepted by the customer?
  - Daily/Frequent builds
    - \* To what extent do automated builds run one or more times everyday?

- Self-managing teams
  - Team Empowerment
    - \* To what extent do the team members determine the amount of work to be done?
    - \* To what extent do the team members take ownership of work items?
    - \* To what extent do the team members hold each other accountable for the work to be completed?
    - \* To what extent do the team members ensure that they complete the work that they are accountable for?
  - Autonomy
    - \* To what extent do the team members determine, plan, and manage their day-to-day activities under reduced or no supervision from the management?
    - \* To what extent do the developers form ad-hoc groups to determine and refine requirements just-in-time?
  - Management support
    - \* To what extent does the management support the self-managing nature of the teams?
- High-bandwidth communication
  - Customer Satisfaction
    - \* To what extent is the product developed so far in-sync with the customers' needs and expectations?
  - Scheduling
    - \* To what extent is the time allocated for the release planning meetings utilized effectively?
    - \* To what extent is the time allocated for the iteration planning meetings utilized effectively?
    - \* To what extent is the time allocated for the retrospective meetings utilized effectively?
    - \* To what extent is the time allocated for the daily progress tracking meetings utilized effectively?
    - \* To what extent do the scheduled meetings (except the daily progress tracking meetings) begin and end on time?
    - \* To what extent do the meetings (except the daily progress tracking meetings) take place as scheduled?
  - Inter- and intra-team communication
    - \* To what extent does open communication prevail between the business and the development team?

- \* To what extent does open communication prevail between the manager and the developers and testers?
  - \* To what extent does open communication prevail between the developers and the testers?
  - \* To what extent does open communication prevail among the developers?
  - \* To what extent does open communication prevail between the external customer/user and the business?
  - \* To what extent does open communication prevail between the external customer/user and the development team?
  - \* To what extent does open communication prevail between members of different teams?
- Client-driven Iterations
  - Requirements Prioritization
    - \* To what extent do the customers establish the priorities of the bug/enhancement?
  - Customer Satisfaction
    - \* To what extent is the product developed so far in-sync with the customers' needs and expectations?
  - Customer Requests
    - \* To what extent are the changes requested by the customers accommodated?
- Short delivery cycles
  - Development time-frames
    - \* To what extent is software released frequently? (length of a release cycle is one year or less)
    - \* To what extent is software released frequently? (length of an iteration is four weeks or less)
  - Customer Satisfaction
    - \* To what extent is the product developed so far in-sync with the customers' needs and expectations?
  - Roll-backs
    - \* To what extent are the deployments not rolled back?
- Iterative Progression
  - Estimation
    - \* To what extent are the estimates for the amount of work to be done during each iteration accurate?

- Iteration length
  - \* To what extent are the iterations timeboxed?
  - \* To what extent is the length of an iteration 4 weeks or less?
- Requirements Management for Iterations
  - \* To what extent is an iteration list maintained?
  - \* To what extent are the bugs/enhancements fully estimated when added to the list?
  - \* To what extent are the bug/enhancement prioritized when added to the list?
- Incremental Development
  - Requirements Management for Releases
    - \* To what extent is a product backlog maintained?
    - \* To what extent are the features prioritized when they are added to the backlog?
    - \* To what extent are the features fully estimated before they are added to the backlog?
  - Timeboxing Releases
    - \* To what extent are the release cycles timeboxed?
    - \* To what extent are only a subset of the identified features developed during a release cycle?
- Evolutionary Requirements
  - Requirements Reprioritization
    - \* To what extent are the features reprioritized as and when new features are identified?
  - Customer Requests
    - \* To what extent are the changes requested by the customers accommodated?
  - Minimal Big Requirements Up Front and Big Design Up Front
    - \* To what extent are only the high level features identified upfront?
    - \* To what extent are the architecture requirements allowed to evolve over time?
- Minimal Documentation
  - Maintaining documentation
    - \* To what extent is minimal documentation supported by teams?
    - \* To what extent is minimal documentation created/developed?
    - \* To what extent is minimal documentation recorded/archived?
    - \* To what extent is minimal documentation maintained?

# B

## Effectiveness Questions for Company F's Teams

**To what rate are the following implemented?**

1. Iterative Progression (Develop the product over several iterations/cycles in sequence)
2. Incremental Development (Create the product incrementally. Develop only a selected/prioritized set of bugs/enhancements during a release cycle)
3. Short Delivery Cycles (Deliver valuable software frequently)
4. Evolutionary Requirements (Allow the features/requirements to evolve over the development lifecycle)
5. Refactoring (Refine the architecture, design, code, and/or other process artifacts regularly to improve the quality of that artifact)
6. Self-Managing Teams (Allow the team members to determine, plan, and manage their day-to-day activities and duties under reduced or no supervision)
7. Continuous Integration (Team members integrate their work frequently; usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible)
8. Minimal Documentation (Maintain just-enough documentation to satisfy the needs of the development team and the customer)
9. High-bandwidth communication (Facilitate continuous communication among the (developers, testers, customers) (in-person, face-to-face interactions))

## *APPENDIX B. EFFECTIVENESS QUESTIONS FOR COMPANY F'S TEAMS*

---

10. Client-driven iterations (The customers and users prioritize the bugs/enhancements. Build only what is of value to the customers and users)
11. Appropriate distribution of expertise (Select the right people to complete the tasks. Ensure that the team is composed of people with the appropriate skill sets to complete the assigned tasks)
12. Configuration Management (Manage the evolution of the product and other artifacts, both during the initial stages of development and during all stages of maintenance)
13. Adherence to Standards (Conform to a set of standards that the team or organization has agreed to comply with, e.g. Coding standards.)

# C

## Perceptive Agile Measurement

- Iteration Planning
  - All members of the technical team actively participated during iteration planning meetings
  - All technical team members took part in defining the effort estimates for requirements of the current iteration
  - When effort estimates differed, the technical team members discussed their underlying assumption
  - All concerns from team members about reaching the iteration goals were considered
  - The effort estimates for the iteration scope items were modified only by the technical team members
  - Each developer signed up for tasks on a completely voluntary basis
  - The customer picked the priority of the requirements in the iteration plan
- Iterative Development
  - We implemented our code in short iterations
  - The team rather reduced the scope than delayed the deadline
  - When the scope could not be implemented due to constraints, the team held active discussions on re-prioritization with the customer on what to finish within the iteration
  - We kept the iteration deadlines
  - At the end of an iteration, we delivered a potentially shippable product
  - The software delivered at iteration end always met quality requirements of production code



- Working software was the primary measure for project progress
- Continuous Integration And Testing
  - The team integrated continuously
  - Developers had the most recent version of code available
  - Code was checked in quickly to avoid code synchronization/integration hassles
  - The implemented code was written to pass the test case
  - New code was written with unit tests covering its main functionality
  - Automated unit tests sufficiently covered all critical parts of the production code
  - For detecting bugs, test reports from automated unit tests were systematically used to capture the bugs
  - All unit tests were run and passed when a task was finished and before checking in and integrating
  - There were enough unit tests and automated system tests to allow developers to safely change any code
- Stand-Up Meetings
  - Stand up meetings were extremely short (max. 15 minutes)
  - Stand up meetings were to the point, focusing only on what had been done and needed to be done on that day
  - All relevant technical issues or organizational impediments came up in the stand up meetings
  - Stand up meetings provided the quickest way to notify other team members about problems
  - When people reported problems in the stand up meetings, team members offered to help instantly
- Customer Access
  - The customer was reachable
  - The developers could contact the customer directly or through a customer contact person without any bureaucratic hurdles
  - The developers had responses from the customer in a timely manner
  - The feedback from the customer was clear and clarified his requirements or open issues to the developers
- Customer Acceptance Tests
  - How often did you apply customer acceptance tests?

- A requirement was not regarded as finished until its acceptance tests (with the customer) had passed
- Customer acceptance tests were used as the ultimate way to verify system functionality and customer requirements
- The customer provided a comprehensive set of test criteria for customer acceptance
- The customer focused primarily on customer acceptance tests to determine what had been accomplished at the end of an iteration
- Retrospectives
  - How often did you apply retrospectives?
  - All team members actively participated in gathering lessons learned in the retrospectives
  - The retrospectives helped us become aware of what we did well in the past iteration(s)
  - The retrospectives helped us become aware of what we should improve in the upcoming iteration(s)
  - In the retrospectives (or shortly afterwards), we systematically assigned all important points for improvement to responsible individuals
  - Our team followed up intensively on the progress of each improvement point elaborated in a retrospective
- Co-Location
  - Developers were located majorly in
  - All members of the technical team (including QA engineers, db admins) were located in
  - Requirements engineers were located with developers in
  - The project/release manager worked with the developers in
  - The customer was located with the developers in

# D

## Team Agility Assessment

- Product Ownership
  - Backlog prioritized and ranked by business value
  - Backlog estimated at gross level
  - Product owner defines acceptance criteria for stories
  - Product owner and stakeholders participate at iteration and release planning
  - Product owner and stakeholders participate at iteration and release review
  - Product owner collaboration with team is continuous
  - Stories sufficiently elaborated prior to planning meetings
- Release Planning and Tracking
  - Release theme established and communicated
  - Release planning meeting attended and effective
  - Release backlog defined
  - Release backlog ranked by priority
  - Release backlog estimated at plan level
  - The team has small and frequent releases
  - The team has a common language and metaphor to describe the release
  - Release progress tracked by feature acceptance
  - Team completes and product owner accepts the release by the release date
  - Release review meeting attended and effective
  - Team inspects and adapts (continuous improvement) the release plan
  - Team meets its commitments to release

- Iteration Planning and Tracking
  - Iteration theme established and communicated
  - Iteration planning meeting attended and effective
  - Team velocity measured and used for planning
  - Iteration backlog defined
  - Iteration backlog ranked by priority
  - Team develops and manages iteration backlog
  - Team defines, estimates, and selects their own work (stories and tasks)
  - Team discusses acceptance criteria during iteration planning
  - Team manages interdependencies and constraints
  - Iteration progress tracked by task to do (burn-down chart) and card acceptance (velocity)
  - Work is not added by the product owner during the iteration
  - Team completes and product owner accepts the iteration
  - Iterations are of a consistent fixed length
  - Iterations are no more than four weeks in length
  - Iteration review meeting attended and effective
  - Team inspects and adapts (continuous improvement) the Iteration Plan
- Team
  - The whole team is present at release planning meetings
  - Team is cross-functional with integrated product owner, development, documentation and QA
  - Team is colocated
  - Team is 100% dedicated to the release (no time-slicing)
  - Team is smaller than 15 people
  - Team works in a physical environment that fosters collaboration
  - Team works at a sustainable pace
  - Team members complete commitments
  - Daily standup on time, fully attended and effectively communicates
  - Team leads communication; communication not managed
  - Team self-polices and reinforces use of agile practices and rules
  - Team inspects and adapts (continuous improvement) the overall process
  - Team Coach/Scrum Master exists, is full-time, and is effective
  - The team has an effective channel for obstacle escalation

- Testing Practices
  - All testing is done within the iteration and does not lag behind
  - Iteration defects are fixed within that iteration
  - Unit tests written before development
  - Acceptance tests written before development
  - 100% automated unit test coverage
  - Automated acceptance tests
- Development Practices/Infrastructure
  - Source control system exists
  - Continuous build with 100% successful builds
  - Developers integrate code multiple times per day
  - Team has administrative access to their own workstations
  - Team has administrative control over their development environment
  - Team is permitted to refactor anywhere in the code base
  - Adequate and effective code review practices
  - Coding standards exist and applied
  - Stories accepted and demonstrated on integrated build
  - Refactoring is continuous
  - Pair programming is practiced
  - Identical builds for developers' workstations

# Bibliography

- [1] Agility, C., ????  
URL <http://comparativeagility.com>
- [2] Ambler, S. W., 2011. “Has agile peaked?”.
- [3] Ambysoft, 2013. “How agile are you? 2013 survey”.  
URL <http://www.ambysoft.com/surveys/howAgileAreYou2013.html#Figure6>
- [4] Arthur, J., Nance, R., December 1990. A framework for assessing the adequacy and effectiveness of software development methodologies. In: Proceedings of the Fifteenth Annual Software Engineering Workshop. Greenbelt, MD.
- [5] Aveling, B., 2004. Xp lite considered harmful? In: Eckstein, J., Baumeister, H. (Eds.), Extreme Programming and Agile Processes in Software Engineering. Vol. 3092. Springer Berlin Heidelberg, pp. 94–103.
- [6] Beck, K., Andres, C., 2004. Extreme Programming Explained: Embrace Change (2Nd Edition). Addison-Wesley Professional.
- [7] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D., 2001. Manifesto for agile software development.  
URL <http://www.agilemanifesto.org/>
- [8] Boehm, B., Turner, R., June 2003. Observations on balancing discipline and agility. In: Agile Development Conference, 2003. ADC 2003. Proceedings of the. pp. 32–39.
- [9] Boehm, B., Turner, R., May 2004. Balancing agility and discipline: evaluating and integrating agile and plan-driven methods. In: Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on. pp. 718–719.
- [10] Cockburn, A., 2002. Agile software development. Agile software development series. Addison-Wesley.  
URL <http://books.google.se/books?id=JxYQ1Zb61zkC>

- [11] Cockburn, A., 2004. Crystal Clear a Human-powered Methodology for Small Teams, 1st Edition. Addison-Wesley Professional.
- [12] Datta, S., 2009. Metrics and techniques to guide software development. Ph.D. thesis, Florida State University College of Arts and Sciences.
- [13] Escobar-Sarmiento, V., Linares-Vasquez, M., Oct 2012. A model for measuring agility in small and medium software development enterprises. In: Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En. pp. 1–10.
- [14] Highsmith, III, J. A., 2000. Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Dorset House Publishing Co., Inc., New York, NY, USA.
- [15] Hossain, E., Ali Babar, M., Verner, J., 2009. Towards a framework for using agile approaches in global software development. In: Product-Focused Software Process Improvement. Vol. 32. Springer Berlin Heidelberg, pp. 126–140.
- [16] Ikoma, M., Ooshima, M., Tanida, T., Oba, M., Sakai, S., May 2009. Using a validation model to measure the agility of software development in a large software development organization. In: Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on. pp. 91–100.
- [17] Jalali, S., 2012. Efficient software development through agile methods. Ph.D. thesis, Blekinge Institute of Technology.
- [18] Koch, A., 2005. Agile Software Development: Evaluating The Methods For Your Organization. Artech House computing library. Artech House, Incorporated.  
URL <http://books.google.se/books?id=vJ99QgAACAAJ>
- [19] Lappo, P., Andrew, H. C. T., 2004. Assessing agility. Vol. 3092 of Lecture Notes in Computer Science. Springer, pp. 331–338.
- [20] Leffingwell, D., 2007. Scaling Software Agility: Best Practices for Large Enterprises (The Agile Software Development Series). Addison-Wesley Professional.
- [21] Nagel, R. N., Dove, R., 1991. 21st Century Manufacturing Enterprise Strategy: An Industry-Led View. Diane Pub Co.
- [22] Nietzsche, F., 2012. Thus Spoke Zarathustra. Simon & Brown.
- [23] Palmer, S. R., Felsing, M., 2001. A Practical Guide to Feature-Driven Development, 1st Edition. Pearson Education.
- [24] Poonacha, K., Bhattacharya, S., Jan 2012. Towards a framework for assessing agility. In: System Science (HICSS), 2012 45th Hawaii International Conference on. pp. 5329–5338.

- [25] Qumer, A., Henderson-Sellers, B., 2006. Measuring agility and adaptability of agile methods: A 4 dimensional analytical tool.
- [26] Qumer, A., Henderson-Sellers, B., Nov. 2008. A framework to support the evaluation, adoption and improvement of agile methods in practice. *J. Syst. Softw.* 81 (11), 1899–1919.
- [27] Reifer, D. J., 2002. How to get the most out of extreme programming/agile methods. In: *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002*. Springer-Verlag, pp. 185–196.
- [28] Sahota, M., 2012. An Agile Adoption And Transformation Survival Guide. lulu.com.
- [29] Schwaber, K., Beedle, M., 2001. *Agile Software Development with Scrum* (Series in Agile Software Development). Prentice Hall.
- [30] Shawky, D., Ali, A., Nov 2010. A practical measure for the agility of software development processes. In: *Computer Technology and Development (ICCTD), 2010 2nd International Conference on*. pp. 230–234.
- [31] Sidky, A., 2007. A structured approach to adopting agile practices: The agile adoption framework. Ph.D. thesis, Virginia Polytechnic Institute and State University.
- [32] Sidky, A., Arthur, J., Bohner, S., 2007. A disciplined approach to adopting agile practices: the agile adoption framework. *Innovations in systems and software engineering* 3 (3), 203–216.
- [33] So, C., Scholl, W., 2009. Perceptive agile measurement: New instruments for quantitative studies in the pursuit of the social-psychological effect of agile practices. Vol. 31 of *Lecture Notes in Business Information Processing*. Springer, pp. 83–93.
- [34] Soundararajan, S., 2013. Assessing agile methods, investigating adequacy, capability and effectiveness. Ph.D. thesis, Virginia Polytechnic Institute and State University.
- [35] Soundararajan, S., Arthur, J., Balci, O., Aug 2012. A methodology for assessing agile software development methods. In: *Agile Conference (AGILE), 2012*. pp. 51–54.
- [36] Studio, T., ????  
URL <http://www.agileassessments.com>
- [37] Sutherland, J., 2008. The scrum but test.  
URL <http://antoine.vernois.net/scrumbut/?page=intro&lang=en>
- [38] Taromirad, M., Ramsin, R., Oct 2008. Cefam: Comprehensive evaluation framework for agile methodologies. In: *Software Engineering Workshop, 2008. SEW '08. 32nd Annual IEEE*. pp. 195–204.



- [39] Taylor, P., Greer, D., Sage, P., Coleman, G., McDaid, K., Lawthers, I., Corr, R., 2006. Applying an agility/discipline assessment for a small software organisation. In: Product-Focused Software Process Improvement. Vol. 4034. Springer Berlin Heidelberg, pp. 290–304.
- [40] VersionOne, 2013. 8th annual state of agile survey.  
URL <http://stateofagile.versionone.com/>
- [41] Waters, K., 2008. How agile are you?  
URL <http://www.allaboutagile.com/how-agile-are-you-take-this-42-point-test/>
- [42] Williams, L., Krebs, W., Layman, L., Antón, A. I., Abrahamsson, P., 2004. Toward a framework for evaluating extreme programming.
- [43] Williams, L., Rubin, K., Cohn, M., Aug 2010. Driving process improvement via comparative agility assessment. In: Agile Conference (AGILE), 2010. pp. 3–10.