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 March 22, 2014

Contents

List of Tables

List of Figures

1

Introduction

Agile 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 reluctanct 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 [17].

In 2001, 17 developers were gathered to create the agile manifesto [6] 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, Feature Driven Development, Crystal, Scrum and others. Most companies use a tailored methodology by following some processes and practices of the aforementioned ones in order to better suit the companies. Williams et al. [29] reports that rarely all XP practices are exercised in their pure form. Sidky et al. [25] 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 [11] the agile methodologies are easier to misunderstand. Such a case could lead to problems later on in the software development process. Sahota [23] 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 [15] 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 [24] defines the 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 better than a group which uses only pair programming but in a more efficient manner.

Practictioners think that declaring being agile is equally good as being agile. According to a suvrvey from Ambysoft [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 copanies are at odds with core agile values based on the most recent survey by VersionOne [28]. Poonacha and Bhattacharya [20] 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 [20]. If we had to use one word we could state it is a way of *being*. Nietzsche [18] 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. Reasearchers 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 since none of them has been validated [13]. 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 many papers of this field.

2.2 Balancing Discipline and Agility

Boehm and Turner [7] did not come up with a tool to measure agility but rather to balance between agility and discipline. Accoring to them [8] 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 descipline. 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 [7] 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 which are the:

- size of a team working in a project
- criticallity of damage of unexpected defects
- culture on how to balance between chaos an order
- dynamism
- personnel which refers to the extended Cockburn [9] skill rating

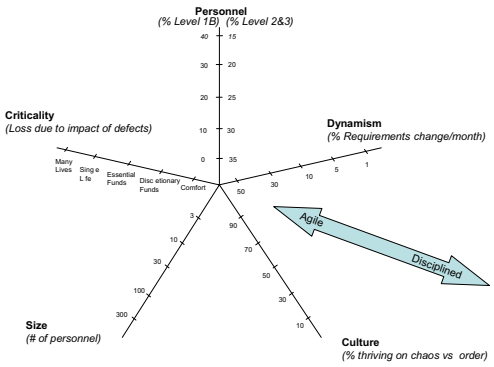

Figure 2.1: Dimensions affecting method selection

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 precedented 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 terittory and the team is considered agile, otherwise it follows a discipline approach.

2.3 4-Dimensional Analytical Tool

Qumer and Henderson-Sellers [22] created the 4-Dimensional Analytical Tool (4-DAT) for analysing and comparing agile methods. 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.

2.3.1 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 [5], Koch [14], Palmer and Felsing [19], Highsmith [12] 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 specifiy?
8. Abstraction Mechanism	Does the method specify abstraction mechanism (object-oriented, agent-oriented)?

Table 2.2: 4-DAT Dimension 1

2.3.2 Dimension 2 - Agility Characterization

The second dimesion is the only quantitavie 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 [22] 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, Phase or Practices)*

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

2.3.3 Dimension 3 - Agile Values Characterization

The third dimension consists of six agile values. Four of them are derived directly from the Agile Manifesto [6], while the fifth comes from [14]. The last value is suggested by Qumer and Henderson-Sellers [22] 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

2.3.4 Dimension 4 - Software Process Characterization

The fourth dimension examines the practices that support four processes as these are presented by Qumer and Henderson-Sellers [22]. Table 2.5 lists these processess.

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

2.4 Leffingwell

Leffingwell [16] 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. The results are representd in a radar chart.

2.5 Escobar

Escobar-Sarmiento and Linares-Vasquez [11] 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 [22]
- Project Agility Assessment - proposed by Taylor et al. [27]
- Workteam Agility Assessment - proposed by Leffingwell [16]
- 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 2.2 one can see the model with a short description about which tool should be used at each level for each stage.

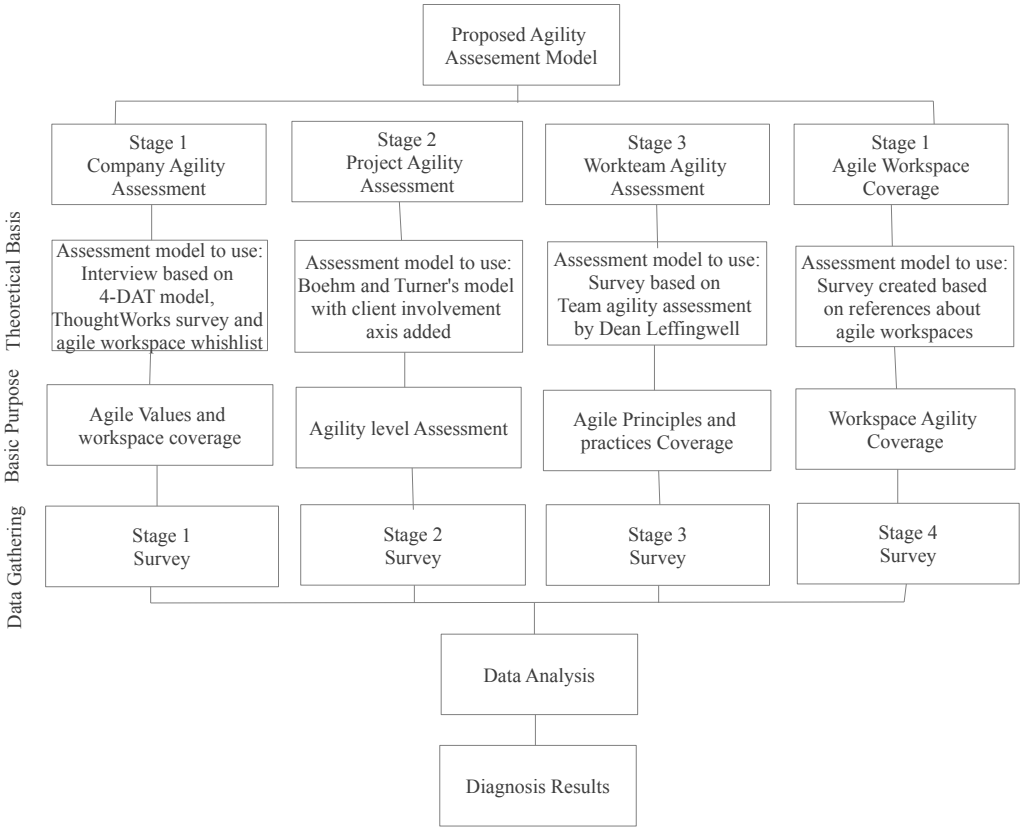

Figure 2.2: Escobar - Vasquez model for assessing agility

2.6 Sidky

2.7 OPS Framework

Soundararajan [26] created the Objectives, Principles and Stategies (OPS) Framework in order to assess the "goodness" of an agile methodology. The focus of this tool is mainly on eXtreme Programming [5], Feature Driven Development (FDD) [19], Lean [21], Crystal [10] 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 [4]
- Principles - what rules a process in order to achieve an objective according to [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 asses 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

The OPS Framework identifies in total 5 objectives, 9 principles, 17 strategies 54 linkages and 80 indicators.

2.8 Thoughtworks

Thoughtworks [1] 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.

2.9 AHP - ANFIS Framework

Poonacha and Bhattacharya [20] created tool for measuring agility they identified 17 parameters grouped in four parameter groups as it can be seen in table (reference to

table). While the last group is an indicator of performance, the first three groups migitate 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 minize 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 Involvment	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

2.10 CEFAM

2.11 Williams - XP-EF

2.12 Validation Model to Measure the Agility

2.13 Other

Taylor et al. [27] modified the tool created by Boehm and Turner [7] 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.

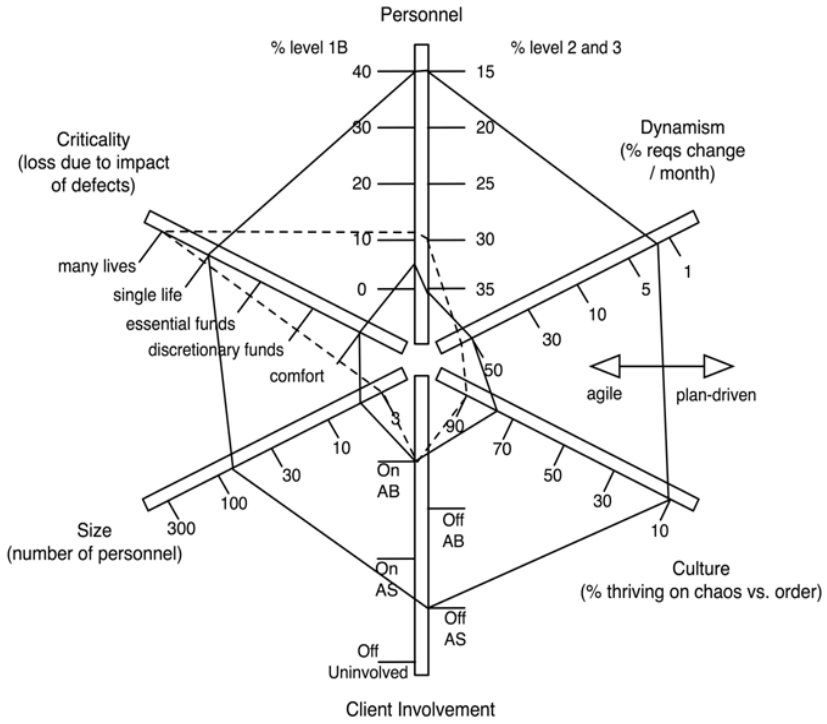

Figure 2.3: ??????????

- 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.

3

Case Study

3.1 Problem

3.2 Company F

For the validation of the OPS Framework company F was willing to participate in order to measure how agile are its teams. Company F [1] is a United States company which acts in the POS [2] 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.

3.2.1 Methodology F

In general, company F does not follow a specific agile methodology, but rather a tailored mix of others which suit the needs of its team.
Methodology F, as we can name it, embraces the following practices from the various agile methodologies [14], some of the them in bigger and some of them in a smaller extent.

[1] F is the first letter of the company's name
[2] Point Of Sales

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

Table 3.1: Practices embraced in method F

3.2.2 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.

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

Table 3.2: Team A - Profile

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

Table 3.3: Team B - Profile

Team Size	4
Roles	Team Leader (1) Developers (1) Testers (2)
Development Process	Method C
Area	Java
Tools used	Perforce Eclipse IDE
Iteration length	2-3 weeks

Table 3.4: Team C - Profile

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

Table 3.5: Team D - Profile

3.3 Method

3.3.1 Introduction

In order to measure the adequacy, the capability and the effectiveness of methodology F, the described method by Soundararajan [26] was followed.

3.3.2 Scales

Answer	**Score**
Yes/Fully	5
Partiallty	3
No/Marginally	1

Table 3.6: Scale for Capability

Answer	**Score**
Maximally	5
Considerably	5
Moderately	3
Somewhat	2
Marginally	1

Table 3.7: Scale for Effectiveness

Score Range	**Interpretation**
4.5 - 5.0	Maximal
3.5 - 4.4	Considerable
2.5 - 3.4	Moderate
1.5 - 2.4	Somewhat
1 - 1.4	Marginal

Table 3.8: Scale for intrepreting final scores

3.3.3 Assessing the Adequacy

In order to assess the adequacy a top down approach was followed for each objective, principle and strategy as it is described by **?**] For the analysis of each objective, principle and strategy the analysis of agile methodologies based on Koch [14] was followed.
Initially the objectives fulfilled by methodology F were identified. As it can be seen in Figure 3.1 all five objectives mandated by the OPS Framework are followed.

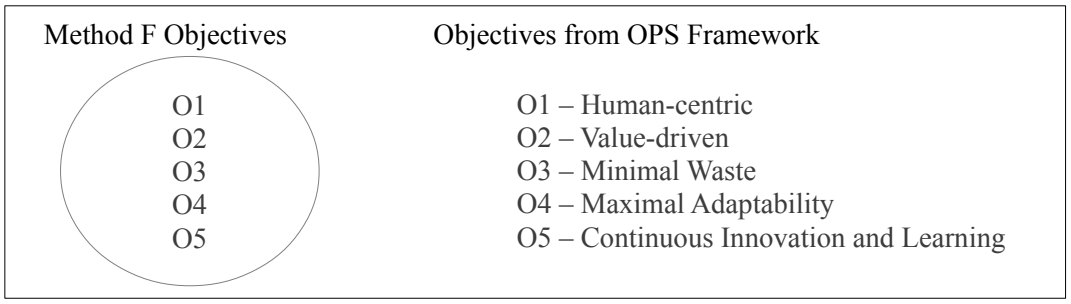

Figure 3.1: Objectives identified in methodology F

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" principles is not entirely followed, but it differs among the teams. Every team is built with motivated individuals, some to a more and some to a lesser extent.

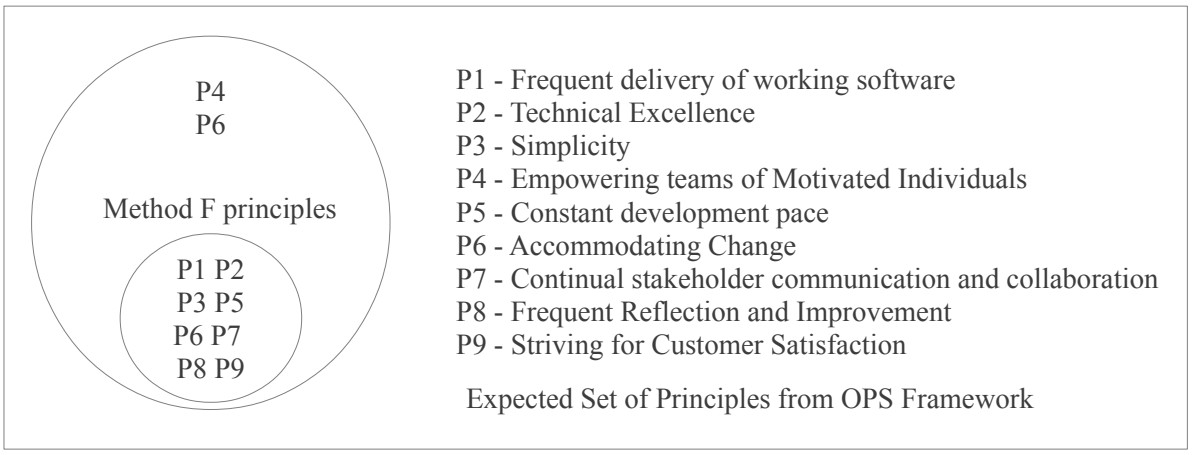

Figure 3.2: Principles identified in methodology F

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 ready 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.

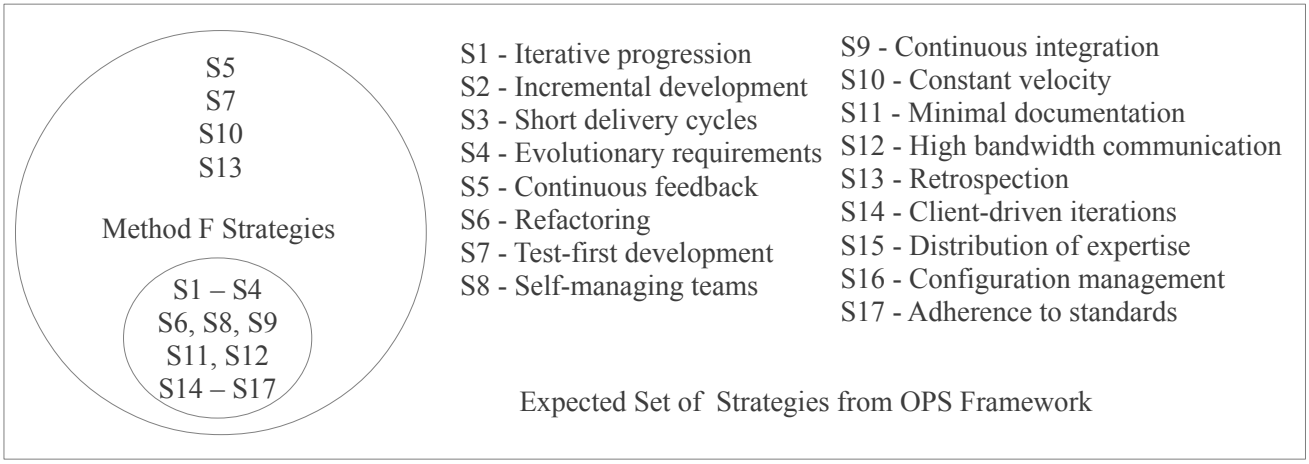

Figure 3.3: Strategies identified in methodology F

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 the methodology is questionnable.

3.3.4 Team A

Capability Assessement

Effectiveness Assessement

3.3.5 Team B

Capability Assessement

Effectiveness Assessement

3.3.6 Team C

Capability Assessement

Effectiveness Assessement

3.3.7 Team D

Capability Assessement

Effectiveness Assessement

3.4 Results

3.5 Discussion

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 priorotized 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))

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 meetigs, 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 bureaucratical 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

Bibliography

[1] , ????
URL http://www.agileassessments.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 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] Beck, K., Andres, C., 2004. Extreme Programming Explained: Embrace Change (2Nd Edition). Addison-Wesley Professional.

[6] 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/

[7] 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.

[8] 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.

[9] Cockburn, A., 2002. Agile software development. Agile software development series. Addison-Wesley.
URL http://books.google.se/books?id=JxYQ1Zb61zkC

[10] Cockburn, A., 2004. Crystal Clear a Human-powered Methodology for Small Teams, 1st Edition. Addison-Wesley Professional.

[11] 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.

[12] Highsmith, III, J. A., 2000. Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Dorset House Publishing Co., Inc., New York, NY, USA.

[13] Jalali, S., 2012. Efficient software development through agile methods. Ph.D. thesis, Blekinge Institute of Technology.

[14] 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`

[15] Lappo, P., Andrew, H. C. T., 2004. Assessing agility. Vol. 3092 of Lecture Notes in Computer Science. Springer, pp. 331–338.

[16] Leffingwell, D., 2007. Scaling Software Agility: Best Practices for Large Enterprises (The Agile Software Development Series). Addison-Wesley Professional.

[17] Nagel, R. N., Dove, R., 1991. 21st Century Manufacturing Enterprise Strategy: An Industry-Led View. Diane Pub Co.

[18] Nietzsche, F., 2012. Thus Spoke Zarathustra. Simon & Brown.

[19] Palmer, S. R., Felsing, M., 2001. A Practical Guide to Feature-Driven Development, 1st Edition. Pearson Education.

[20] 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.

[21] Poppendieck, M., Poppendieck, T., 2003. Lean Software Development: An Agile Toolkit. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[22] Qumer, A., Henderson-Sellers, B., 2006. Measuring agility and adaptibility of agile methods: A 4 dimensional analytical tool.

[23] Sahota, M., 2012. An Agile Adoption And Transformation Survival Guide. lulu.com.

[24] Sidky, A., 2007. A structured approach to adopting agile practices: The agile adoption framework. Ph.D. thesis, Virginia Polytechnic Institute and State University.

[25] 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.

[26] Soundararajan, S., 2013. Assessing agile methods, investigating adequacy, capability and effectiveness. Ph.D. thesis, Virginia Polytechnic Institute and State University.

[27] 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: Münch, J., Vierimaa, M. (Eds.), Product-Focused Software Process Improvement. Vol. 4034 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 290–304.

[28] VersionOne, 2013. 8th annual state of agile survey.
URL `http://stateofagile.versionone.com/`

[29] Williams, L., Krebs, W., Layman, L., Antón, A. I., Abrahamsson, P., 2004. Toward a framework for evaluating extreme programming.