Czech University of Life Sciences Prague
Faculty of Economics and Management
Department of Management

Diploma Thesis
Agile Methods in Project Management

Petr Hradil

! ! !
In this place, please insert
the Diploma Thesis Assignment.
(The original goes into one thesis
and the copy into the other)
! ! !

Declaration
I declare that I have worked on my diploma thesis titled Agile Methods in Project Management by myself and I have used only the sources mentioned at the end of the thesis.

In Prague on date 4/9/2010

Petr Hradil

Acknowledgement
I would like to thank name of the supervisor and all other persons, for their advice and support during my work on this Thesis.

Agilní metody v projektovém řízení

Agile Methods in Project Management

Souhrn
Summary in the Czech language (approximately 15 lines)
Klíčová slova:
Keywords in the Czech language (approximately 10)
Summary
English version of the Summary
Keywords:
Keywords in English language (approximately 10)

Contents

1 Introduction

Traditional project management is a discipline of planning, organizing and managing resources. A project is an endeavor that has to fulfill in given timeframe it´s task within the required time, quality, scope and cost. Traditional project management is driven by a central point. In the beginning of the project the initial plan is set and the project team tries it´s best to fulfill the requirements. Among the most frequent traditional methods that are used to control and manage the project we can name Gantt charts, Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT). Gantt chart was firstly designed by Henry Laurence Gantt around years 1910 and 1915. It is to interesting to mention that he was not probably the first. The first reported tool of this kind is a harmonogram developed by Karol Adamiecki. Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT) were developed in the 1950s under the DuPont Corporation (CPM) and US Navy (PERT).

These methods were used essentially in construction and plant development for number of years. In the turn of the millennium new methods were introduced. These methods mainly deal with the decentralization of control and commitment to the team with regard of communication with the customer. Agile methods were firstly presented in Agile Manifesto. The word agile can be defined as "1) marked by ready ability to move with quick easy grace or 2) having a quick resourceful and adaptable character" (1). "Core to Agile software development is the use of light-but-sufficient rules of project behaviour and the use human and communication-oriented rules" (2).

Adaptive way and to have close cooperation with the customer with high level of changing the scope are main attributes of Agile Project Management. Thus, it is

possible to react to changes by the changing business environment and at the same time maintain effectiveness and efficiency. Generally, the emphasis is put on tacit knowledge and sharing via constant face to face communication.

This paper explains two main Agile methods used in Project Management. First, the SCRUM, iterative and incremental framework of agile project management, developed by Hirotaka Takeuchi and Ikimuro Nonaka in 1986. They compared new product development with rugby match where the whole team tries to go to the distance as a unit, passing the ball back and forth. Lately, the Ken Schwaber and Easel Corporation firstly called the method Scrum.

Second, the Extreme Programming (XP), essentially methodology focused on software development projects. The XP advocates constant level of "releases" in very short development cycles. It is well-known because it uses pair programming (when two programmes sitting side-by-side working on one task).

The survey among 32 professionals was conducted. The questionnaire rate of return was about 30 percent. Population covers mainly Central Europen region. The emphasis concentrated mainly on the success perception. Success factors were defined as time, scope, quality and cost.

Futhermore, the paper contains the case study of the project led by agile method. The main goal of the project was to develop the solution for "Datové schránky", Czech proprietary system of delivering information from government agencies to the enterprises in order to speed up the communication.

2 Objectives of Thesis and Metodology

The objective of this paper is about to explain two most frequent Agile methods in Project Management. Iterative SCRUM and more software development focused Extreme Programming (XP). The aim is explain different aspects of these methods.

Moreover, there is a lack of academic research on this topis in other areas than USA. Some analyses can be found but most of the articles and books about agile project management discuss are written by the inventors and promoting their own and mainly covers the projects going in North America and Western Europe. This paper contains a survey conducted in the area of Central Europe, answering various questions with regards to success perception of the project.

In order to answer the objectives of the paper a literature review was conducted. Concerning the discussion and results of the survey basic statistical methods such as mean were conducted. Furthermore, the crosstab analysis was used in order to find our relations. The Fisher´s Exact test was used to determine whether there is or is not a statiticly significant difference in variables. In order to be able to say the power of relation Cramer´s V is computed. Statistical software SPPSS 17 was used.

3 Literature Overview

3.1 Software Development - Introduction

Software development is a complex endeavor. Of course, this news isn't very surprising because the universe is full of complexity. Most complexities we don't know about, and others we are content to leave unexamined. Some like the complex process by which pressure turns coal into diamonds take care of themselves. Others for example, commuting to work every day can tolerate some imprecision. However, it is impossible to ignore complexity in software development. Its results are ephemeral, consisting merely of signals that control machines. The software development process is entirely intellectual, and all of its intermediate products are marginal representations of the thoughts involved. The materials that we use to create the end product are extremely volatile: user requirements for a program the users have yet to see, the interoperation of other programs' signals with the program in question, and the interaction of the most complex organisms on the planet: people. (2)

3.2 Empirical Process Control

Complex problems are those that behave unpredictably. Not only are these problems unpredictable, but even the ways in which they will prove unpredictable are impossible to predict. To put that another way, a statistical sample of the operation of these processes will never yield meaningful insight into their underlying mathematical model, and attempts to create a sample can only be made by summarizing their operation to such a degree of coarseness as to be irrelevant to those trying to understand or manage these processes.

Much of our society is based on processes that work only because their degree of imprecision is acceptable. Wheels wobble, cylinders shake, and brakes jitter, but

this all occurs at a level that doesn't meaningfully impede our use of a car. When we build cars, we fit parts together with a degree of precision fit for their intended purpose. We can manage many processes because the accuracy of the results is limited by our physical perceptions. For example, when I build a cabinet, I need only cut and join the materials with enough precision to make them acceptable to the human eye; if I were aiming only for functionality, I could be far less precise.

What happens when we are building something that requires a degree of precision higher than that obtainable through averaging? What happens if any process that we devise for building cars is too imprecise for our customers, and we need to increase the level of precision? In those cases, we have to guide the process step by step, ensuring that the process converges on an acceptable degree of precision. In cases where convergence doesn't occur, we have to make adaptations to bring the process back into the range of acceptable precision levels. Laying out a process that repeatably will produce acceptable quality output is called *defined process control*. When defined process control cannot be achieved because of the complexity of the intermediate activities, something called empirical process control has to be employed.

It is typical to adopt the defined (theoretical) modeling approach when the underlying mechanisms by which a process operates are reasonably well understood. When the process is too complicated for the defined approach, the empirical approach is the appropriate choice.

—B. A. Ogunnaike and W. H. Ray,
Process Dynamics, Modeling, and Control

We use defined processes whenever possible because with them we can crank up unattended production to such a quantity that the output can be priced as a

commodity. However, if the commodity is of such unacceptable quality as to be unusable, the rework is too great to make the price acceptable, or the cost of unacceptably low yields is too high, we have to turn to and accept the higher costs of empirical process control. In the long run, making successful products the first time using empirical process control turns out to be much cheaper than reworking unsuccessful products using defined process control. There are three legs that hold up every implementation of empirical process control: *visibility*, *inspection*, and *adaptation*. Visibility means that those aspects of the process that affect the outcome must be visible to those controlling the process. Not only must these aspects be visible, but what is visible must also be true. There is no room for deceiving appearances in empirical process control. What does it mean, for example, when someone says that certain functionality is labeled "done"? In software development, asserting that functionality is done might lead someone to assume that it is cleanly coded, refactored, unittested, built, and acceptance-tested. Someone else might assume that the code has only been built. It doesn't matter whether it is visible that this functionality is done if no one can agree what the word "done" means.

The second leg is inspection. The various aspects of the process must be inspected frequently enough that unacceptable variances in the process can be detected. The frequency of inspection has to take into consideration that processes are changed by the very act of inspection. Interestingly, the required frequency of inspection often exceeds the tolerance to inspection of the process. Fortunately, this isn't usually true in software development. The other factor in inspection is the inspector, who must possess the skills to assess what he or she is inspecting.

The third leg of empirical process control is adaptation. If the inspector determines from the inspection that one or more aspects of the process are

outside acceptable limits and that the resulting product will be unacceptable, the inspector must adjust the process or the material being processed. The adjustment must be made as quickly as possible to minimize further deviation.

Let's take code review as an example of an empirical process control. The code is reviewed against coding standards and industry best practices. Everyone involved in the review fully and mutually understands these standards and best practices. The code review occurs whenever someone feels that a section of code or code representing a piece of functionality is complete. The most experienced developers review the code, and their comments and suggestions lead to the developer adjusting his or her code. (3)

3.3 Complexity

Anything can be complex. When complex things interact, the level of complexity goes through the roof. I've limited my enumeration of complexity in software development to the three most significant dimensions: requirements, technology, and people.

It is possible to have simple software requirements. A single customer who is the only person who will use the system can spend enough time with the developer that the two can agree exactly what to build. Assuming that this customer dies immediately after imparting his or her requirements, the requirements will remain constant, and there will be no changes, revisions, or last- minute modifications. More commonly, there are many *stakeholders* (those with an interest in the software and how it works) who have different needs and whose needs frequently change and are difficult to articulate. In most cases, these customers only really start to understand what they want when they are provided with someone else's impression of what they want. Theirs are complex

requirements because their requirements are not only ambiguous, but also constantly changing.

Simple technology exists, but it is rarely used in software development. One might define software development projects as the application of advanced, often unreliable technology to solve business problems and achieve competitive advantage. To compound the complexity of technology, more than one piece is usually employed, and the interfaces of the many are far more complex than the complexity within any single piece.

In Figure 1, the vertical axis traces requirements complexity, and the horizontal axis traces technology complexity. The intersection of these two kinds of complexity defines the total level of complexity of the project. Almost all of today's software development projects are complex. Those that are chaotic are unworkable, and some of their complexities must be resolved before work can progress.

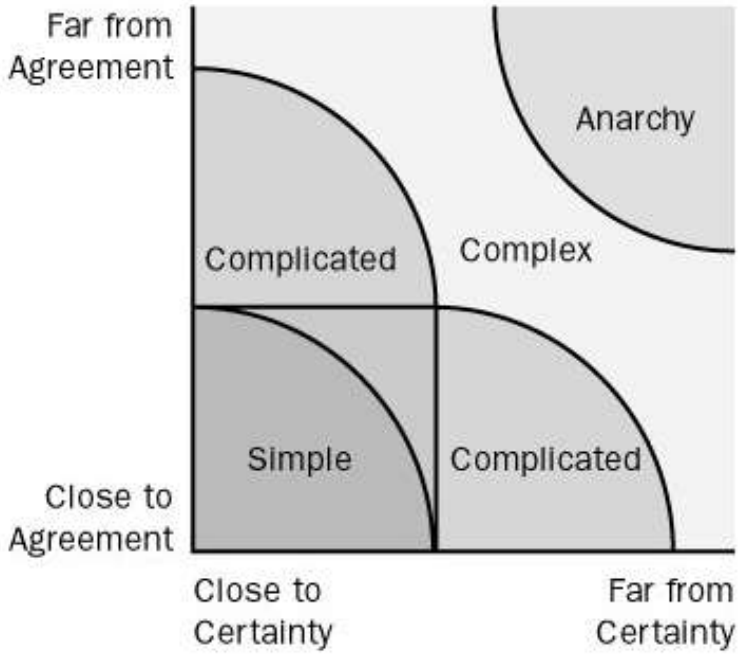

Figure 1 - Complexity assessment graph (3)

The third dimension of complexity is the people developing the software. They all have different skills, intelligence levels, experience, viewpoints, attitudes, and prejudices. Every morning, each wakes up in a different mood than the day before, depending on his or her sleep, health, weather, neighbors, and families. These people then start to work together, and the complexity level goes through the roof. (3)

3.4 Scrum Method

The more complex the system, the more likely it is that central control systems will break down. This is the reason companies decentralize and governments deregulate relinquishing control to independent agents is a time- honored approach to dealing with complexity. Scrum travels this well-trodden path by moving control from a central scheduling and dispatching authority to the individual teams doing the work. The more complex the project, the more necessary it becomes to delegate decision making to independent agents who are close to the work.

Scrum turns small teams into managers of their own fate. We know that when we are responsible for choosing our own driving route from Prague to Brno, we will find a way to get there. We will detour around construction and avoid rush hour traffic jams, making decisions on the fly, adapting to the independent decisions of all of the other drivers out there. Similarly, Scrum Teams accept a challenge and then figure out how to meet that challenge, detouring around roadblocks in creative ways that could not be planned by a central control and dispatching center.

Common sense is a combination of experience, training, humility, wit, and intelligence. People employing Scrum apply common sense every time they find the work is veering off the path leading to the desired results. Yet most of us are

so used to using prescriptive processes—those that say “do this, then do that, and then do this” that we have learned to disregard our common sense and instead await instructions.

Most people responsible for managing projects have been taught a deterministic approach to project management that uses detailed plans, Gantt charts, and work schedules. Scrum is the exact opposite. Unlike these tools, which practically fight against a project’s natural momentum, Scrum shows management how to guide a project along its optimal course, which unfolds as the project proceeds. It is well known that traveling along a learning curve starts from a point where you have to think everything through step by step and ends at a point where you can perform the work in question unconsciously. This is particularly true of Scrum because those steeped in traditional management practices have to unlearn many of them. (3)

3.4.1 The Scrum Introduction

A Scrum project starts with a vision of the system to be developed. The vision might be vague at first, perhaps stated in market terms rather than system terms, but it will become clearer as the project moves forward. The Product Owner is responsible to those funding the project for delivering the vision in a manner that maximizes their ROI. The Product Owner formulates a plan for doing so that includes a Product Backlog. The Product Backlog is a list of functional and nonfunctional requirements that, when turned into functionality, will deliver this vision. The Product Backlog is prioritized so that the items most likely to generate value are top priority and is divided into proposed releases. The prioritized Product Backlog is a starting point, and the contents, priorities, and grouping of the Product Backlog into releases usually changes the moment the project starts—as should be expected. Changes in the Product Backlog reflect

changing business requirements and how quickly or slowly the Team can transform Product Backlog into functionality. (4)

All work is done in Sprints. Each Sprint is an iteration of 30 consecutive calendar days. Each Sprint is initiated with a Sprint planning meeting, where the Product Owner and Team get together to collaborate about what will be done for the next Sprint. Selecting from the highest priority Product Backlog, the Product Owner tells the Team what is desired, and the Team tells the Product Owner how much of what is desired it believes it can turn into functionality over the next Sprint. Sprint planning meetings cannot last longer than eight hours that is, they are time-boxed to avoid too much hand-wringing about what is possible. The goal is to get to work, not to think about working.

The Sprint planning meeting has two parts. The first four hours are spent with the Product Owner presenting the highest priority Product Backlog to the Team. The Team questions him or her about the content, purpose, meaning, and intentions of the Product Backlog. When the Team knows enough, but before the first four hours elapses, the Team selects as much Product Backlog as it believes it can turn into a completed increment of potentially shippable product functionality by the end of the Sprint. The Team commits to the Product Owner that it will do its best. During the second four hours of the Sprint planning meeting, the Team plans out the Sprint. Because the Team is responsible for managing its own work, it needs a tentative plan to start the Sprint. The tasks that compose this plan are placed in a Sprint Backlog; the tasks in the Sprint Backlog emerge as the Sprint evolves. At the start of the second four-hour period of the Sprint planning meeting, the Sprint has started, and the clock is ticking toward the 30-day Sprint time-box.

Every day, the team gets together for a 15-minute meeting called a Daily Scrum. At the Daily Scrum, each Team member answers three questions: What have you done on this project since the last Daily Scrum meeting? What do you plan on doing on this project between now and the next Daily Scrum meeting? What impediments stand in the way of you meeting your commitments to this Sprint and this project? The purpose of the meeting is to synchronize the work of all Team members daily and to schedule any meetings that the Team needs to forward its progress.

At the end of the Sprint, a Sprint review meeting is held. This is a four-hour, time-boxed meeting at which the Team presents what was developed during the Sprint to the Product Owner and any other stakeholders who want to attend. This informal meeting at which the functionality is presented is intended to bring people together and help them collaboratively determined what the Team should do next. After the Sprint review and prior to the next Sprint planning meeting, the ScrumMaster holds a Sprint retrospective meeting with the Team. At this three-hour, time-boxed meeting, the ScrumMaster encourages the Team to revise, within the Scrum process framework and practices, its development process to make it more effective and enjoyable for the next Sprint. Together, the Sprint planning meeting, the Daily Scrum, the Sprint review, and the Sprint retrospective constitute the empirical inspection and adaptation practices of Scrum. Take a look at Figure 2 to see a diagram of the Scrum process.

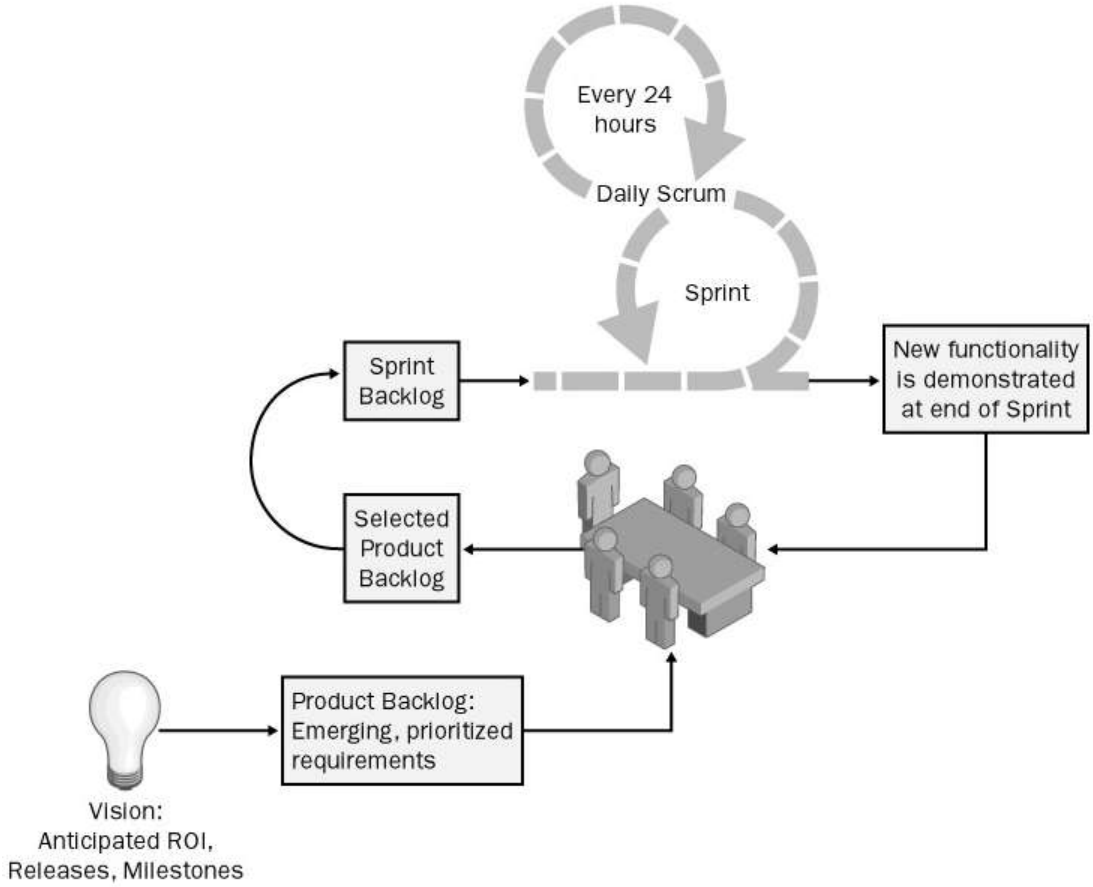

Figure 2 -Scrum Process (3)

3.4.2 The Skeleton and Heart of Scrum

Scrum hangs all of its practices on an iterative, incremental process skeleton. Scrum's skeleton is shown in Figure 3. The lower circle represents an iteration of development activities that occur one after another. The output of each iteration is an increment of product. The upper circle represents the daily inspection that occurs during the iteration, in which the individual team members meet to inspect each others' activities and make appropriate adaptations. Driving the iteration is a list of requirements. This cycle repeats until the project is no longer funded.

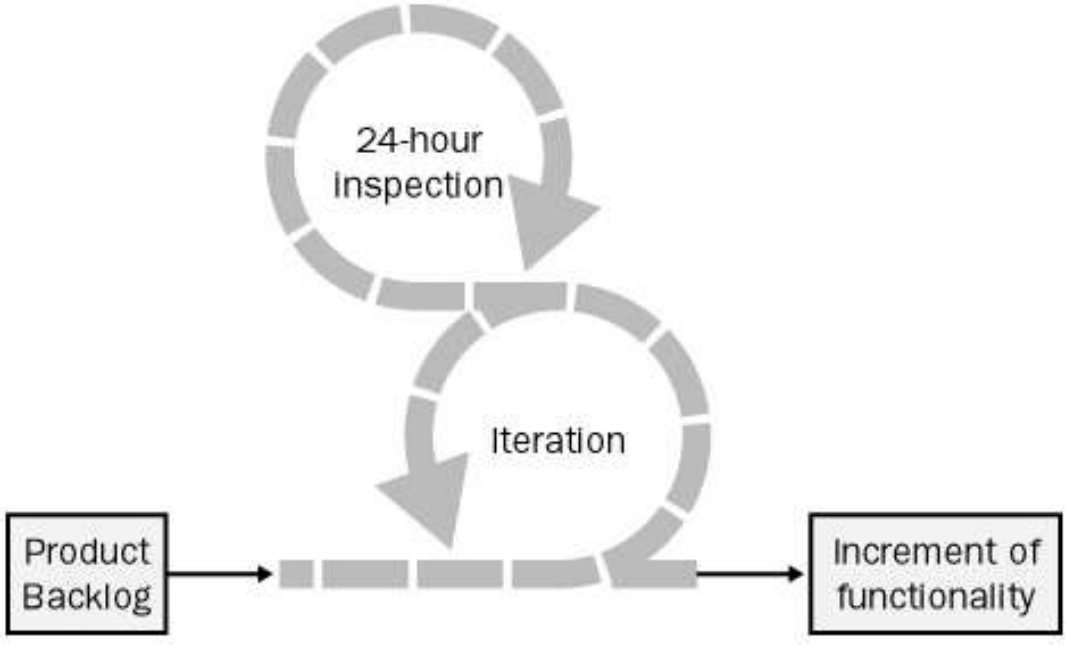

Figure 3 - Scrum skeleton (3)

The skeleton operates this way: At the start of an iteration, the team reviews what it must do. It then selects what it believes it can turn into an increment of potentially shippable functionality by the end of the iteration. The team is then left alone to make its best effort for the rest of the iteration. At the end of the iteration, the team presents the increment of functionality it built so that the stakeholders can inspect the functionality and timely adaptations to the project can be made.

The heart of Scrum lies in the iteration. The team takes a look at the requirements, considers the available technology, and evaluates its own skills and capabilities. It then collectively determines how to build the functionality, modifying its approach daily as it encounters new complexities, difficulties, and surprises. The team figures out what needs to be done and selects the best way to do it. This creative process is the heart of the Scrum's productivity. (5)

3.4.3 Scrum Roles

There are only three Scrum roles: the Product Owner, the Team, and the ScrumMaster. All management responsibilities in a project are divided among these three roles. The Product Owner is responsible for representing the interests of everyone with a stake in the project and its resulting system. The Product Owner achieves initial and ongoing funding for the project by creating the project's initial overall requirements, return on investment (ROI) objectives, and release plans. The list of requirements is called the Product Backlog. The Product Owner is responsible for using the Product Backlog to ensure that the most valuable functionality is produced first and built upon; this is achieved by frequently prioritizing the Product Backlog to queue up the most valuable requirements for the next iteration. The Team is responsible for developing functionality. Teams are self-managing, self-organizing, and cross-functional, and they are responsible for figuring out how to turn Product Backlog into an increment of functionality within an iteration and managing their own work to do so. Team members are collectively responsible for the success of each iteration and of the project as a whole. The ScrumMaster is responsible for the Scrum process, for teaching Scrum to everyone involved in the project, for implementing Scrum so that it fits within an organization's culture and still delivers the expected benefits, and for ensuring that everyone follows Scrum rules and practices.

The people who fill these roles are those who have committed to the project. Others might be interested in the project, but they aren't on the hook. Scrum makes a clear distinction between these two groups and ensures that those who are responsible for the project have the authority to do what is necessary for its success and that those who aren't responsible can't interfere unnecesarily. (3)

3.5 Scrum Method Describtion

3.5.1 Sprint Plannig Meeting

The Sprint planning meeting is time-boxed to 8 hours and consists of two segments that are time-boxed to 4 hours each. The first segment is for selecting Product Backlog; the second segment is for preparing a Sprint Backlog.

- The attendees are the ScrumMaster, the Product Owner, and the Team. Additional parties can be invited by any of these people to provide additional business domain or technology domain information and advice, but they are dismissed after this information is provided. There are no chickens as observers.
- The Product Owner must prepare the Product Backlog prior to the meeting. In the absence of either the Product Owner or the Product Backlog, the ScrumMaster is required to construct an adequate Product Backlog prior to the meeting and to stand in for the Product Owner.
- The goal of the first segment, or first 4 hours, is for the Team to select those Product Backlog items that it believes it can commit to turning into an increment of potentially shippable product functionality. The Team will demonstrate this functionality to the Product Owner and stakeholders at the Sprint review meeting at the end of the Sprint.
- The Team can make suggestions, but the decision of what Product Backlog can constitute the Sprint is the responsibility of the Product Owner.
- The Team is responsible for determining how much of the Product Backlog that the Product Owner wants worked on the Team will attempt to do during the Sprint.

- Time-boxing the first segment to 4 hours means that this is all of the time that is available for analyzing the Product Backlog. Further analysis must be performed during the Sprint. Large-grained, high-priority Product Backlog with imprecise estimates might not be thoroughly understood during this part of the Sprint planning meeting and might result in the Team not being able to complete all of the Product Backlog that it selects.
- The second segment of the Sprint Planning meeting occurs immediately after the first segment and is also time-boxed to 4 hours.
- The Product Owner must be available to the Team during the second segment to answer questions that the Team might have about the Product Backlog.
- It is up to the Team, acting solely on its own and without any direction from outside the Team, to figure out during the second segment how it will turn the selected Product Backlog into an increment of potentially shippable product functionality. No one else is allowed to do anything but observe or answer questions seeking further information.
- The output of the second segment of the Sprint planning meeting is a list, called the Sprint Backlog, of tasks, task estimates, and assignments that will start the Team on the work of developing the functionality. The task list might not be complete, but it must be complete enough to reflect mutual commitment on the part of all Team members and to carry them through the first part of the Sprint, while the Team devises more tasks in the Sprint Backlog. (6)

3.5.2 Daily Scrum Meeting

The Daily Scrum meeting is time-boxed to 15 minutes regardless of the number of Team members.

- Hold the Daily Scrum in the same place at the same time every work day. The Daily Scrum is best held first thing in the day so that the first thing Team members do on arriving at work is think of what they did the day before and what they plan to do today.
- All Team members are required to attend. If for some reason a Team member can't attend in person, the absent member must either attend by telephone or by having another Team member report on the absent member's status.
- Team members must be prompt. The ScrumMaster starts the meeting at the appointed time, regardless of who is present. Any members who are late pay $1 to the ScrumMaster immediately.
- The ScrumMaster begins the meeting by starting with the person immediately to his or her left and proceeding counterclockwise around the room until everyone has reported.
- Each Team member should respond to three questions only:
    - What have you done since the last Daily Scrum regarding this project?
    - What will you do between now and the next Daily Scrum meeting regarding this project?
    - What impedes you from performing your work as effectively as possible?
- Team members should not digress beyond answering these three questions into issues, designs, discussion of problems, or gossip. The ScrumMaster is responsible for moving the reporting along briskly, from person to person.

- During the Daily Scrum, only one person talks at a time. That person is the one who is reporting his or her status. Everyone else listens. There are no side conversations.
- When a Team member reports something that is of interest to other Team members or needs the assistance of other Team members, any Team member can immediately arrange for all interested parties to get together after the Daily Scrum to set up a meeting.
- Chickens are not allowed to talk, make observations, make faces, or otherwise make their presence in the Daily Scrum meeting obtrusive.
- Chickens stand on the periphery of the Team so as not to interfere with the meeting.
- If too many chickens attend the meeting, the ScrumMaster can limit attendance so that the meeting can remain orderly and focused.
- Chickens are not allowed to talk with Team members after the meeting for clarification or to provide advice or instructions.
- Pigs or chickens who cannot or will not conform to the above rules can be excluded from the meeting (chickens) or removed from the Team (pigs). (3)

3.5.3 The Sprint

The Sprint is time-boxed to 30 consecutive calendar days. Aside from other factors, this is the amount of time required for a Team to build something of significant interest to the Product Owner and stakeholders and bring it to a state where it is potentially shippable. This is also the maximum time that can be allocated without the Team doing so much work that it requires artifacts and documentation to support its thought processes. It is also the maximum time that

most stakeholders will wait without losing interest in the Team's progress and without losing their belief that the Team is doing something meaningful for them.

- The Team can seek outside advice, help, information, and support during the Sprint.
- No one can provide advice, instructions, commentary, or direction to the Team during the Sprint. The Team is utterly self-managing.
- The Team commits to Product Backlog during the Sprint planning meeting. No one is allowed to change this Product Backlog during the Sprint. The Product Backlog is frozen until the end of the Sprint.
- If the Sprint proves to be not viable, the ScrumMaster can abnormally terminate the Sprint and initiate a new Sprint planning meeting to initiate the next Sprint. The ScrumMaster can make this change of his or her own accord or as requested by the Team or the Product Owner. The Sprint can prove to be not viable if the technology proves unworkable, if the business conditions change so that the Sprint will not be of value to the business, or if the Team is interfered with during the Sprint by anyone outside the Team.
- If the Team feels itself unable to complete all of the committed Product Backlog during the Sprint, it can consult with the Product Owner on which items to remove from the current Sprint. If so many items require removal that the Sprint has lost its value and meaning, the ScrumMaster can abnormally terminate the Sprint, as previously stated.

If the Team determines that it can address more Product Backlog during the Sprint than it selected during the Sprint planning meeting, it can consult with the Product Owner on which additional Product Backlog items can be added to the Sprint. (3)

3.5.4 Sprint Review Meeting

The Sprint review meeting is time-boxed to 4 hours.

- The Team should not spend more than 1 hour preparing for the Sprint review.
- The purpose of the Sprint review is for the Team to present to the Product Owner and stakeholders functionality that is done. Although the meaning of “done” can vary from organization to organization, it usually means that the functionality is completely engineered and could be potentially shipped or implemented. If “done” has another meaning, make sure that the Product Owner and stakeholders understand it.
- Functionality that isn’t “done” cannot be presented.
- Artifacts that aren’t functionality cannot be presented except when used in support of understanding the demonstrated functionality. Artifacts cannot be shown as work products, and their use must be minimized to avoid confusing stakeholders or requiring them to understand how systems development works.
- Functionality should be presented on the Team member workstations and executed from the server closest to production—usually a quality assurance (QA) environment server.
- The Sprint review starts with a Team member presenting the Sprint goal, the Product Backlog committed to, and the Product Backlog completed. Different Team members can then discuss what went well and what didn’t go well in the Sprint.
- The majority of the Sprint review is spent with Team members presenting functionality, answering stakeholder questions regarding the presentation, and noting changes that are desired.

- At the end of the presentations, the stakeholders are polled, one by one, to get their impressions, any desired changes, and the priority of these changes.
- The Product Owner discusses with the stakeholders and the Team potential rearrangement of the Product Backlog based on the feedback.
- Stakeholders are free to voice any comments, observations, or criticisms regarding the increment of potentially shippable product functionality between presentations.
- Stakeholders can identify functionality that wasn't delivered or wasn't delivered as expected and request that such functionality be placed in the Product Backlog for prioritization.
- Stakeholders can identify any new functionality that occurs to them as they view the presentation and request that the functionality be added to the Product Backlog for prioritization.
- The ScrumMaster should attempt to determine the number of people who expect to attend the Sprint review meeting and set up the meeting to accommodate them.
- At the end of the Sprint review, the ScrumMaster announces the place and date of the next Sprint review to the Product Owner and all stakeholders. (3)

3.5.5 The Sprint Retrospective Meeting

The Sprint retrospective meeting is time-boxed to 3 hours.

- It is attended only by the Team, the ScrumMaster, and the Product Owner. The Product Owner is optional.
- Start the meeting by having all Team members answer two questions:

    - What went well during the last Sprint?
    - What could be improved in the next Sprint?
- The ScrumMaster writes down the Team's answers in summary form.
- The Team prioritizes in which order it wants to talk about the potential improvements.
- The ScrumMaster is not at this meeting to provide answers, but to facilitate the Team's search for better ways for the Scrum process to work for it.
- Actionable items that can be added to the next Sprint should be devised as high-priority nonfunctional Product Backlog. Retrospectives that don't result in change are sterile and frustrating. (3)

3.5.6 Product Backlog

The requirements for the system or product being developed by the project(s) are listed in the Product Backlog. The Product Owner is responsible for the contents, prioritization, and availability of the Product Backlog. The Product Backlog is never complete, and the Product Backlog used in the project plan is merely an initial estimate of the requirements. The Product Backlog evolves as the product and the environment in which it will be used evolves. The Product Backlog is dynamic; management constantly changes it to identify what the product needs to be appropriate, competitive, and useful. As long as a product exists, the Product Backlog also exists.

Backlog Description	Initial Estimate	Adjust-ment Factor	Adjusted Estimate	work remaining until completion						
				1	2	3	4	5	6	7
Title Import				256	209	193	140	140	140	140
Project selection or new	3	0.2	3.6	3.6	0	0	0	0	0	0
Template backlog for new projects	2	0.2	2.4	2.4	0	0	0	0	0	0
Create product backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Create sprint backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Display tree view of product backlog, releases, sprints	2	0.2	2.4	2.4	0	0	0	0	0	0
Sprint-1	13	0.2	15.6	16	0	0	0	0	0	0
Create a new window containing product backlog template	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Create a new window containing sprint backlog template	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Burndown window of product backlog	5	0.2	6	6	6	0	0	0	0	0
Burndown window of sprint backlog	1	0.2	1.2	1.2	1.2	0	0	0	0	0
Display tree view of product backlog, releases, prints	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Display burndown for selected sprint or release	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Sprint-2	16	0.2	19.2	19	19	1.2	0	0	0	0
Automatic recalculating of values and totals	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
As changes are made to backlog in secondary window, update burndown graph on main page	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Hide/automatic redisplay of burndown window	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
Insert Sprint capability ... adds summing Sprint row	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Insert Release capability ... adds summary row for backlog in Sprint	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Owner/assigned capability and columns optional	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Print burndown graphs	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Sprint-3	14	0.2	16.8	17	17	17	0	0	0	0
Duplicate incomplete backlog without affecting totals	5	0.2	6	6	6	6	6	6	6	6
Note capability	6	0.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2
What-if release capability on burndown graph	15	0.2	18	18	18	18	18	18	18	18
Trend capability on burndown server	2	0.2	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4
Publish facility for entire project, publishing it as HTML web pages	11	0.2	13.2	0	0	13	13	13	13	13

Figure 4 - Product Backlog (3)

The first four columns are the Product Backlog item name, the initial estimate, the complexity factor, and the adjusted estimate. The complexity factor increases the estimate due to project characteristics that reduce the productivity of the Team. The remaining columns represent the Sprints during which the Product Backlog is developed. When the Product Backlog is first thought of and entered, its estimated work is placed into the column of the Sprint that is going on at that time. The developers devised most of the backlog items shown before starting this project.

A *burndown chart* shows the amount of work remaining across time. The burndown chart is an excellent way of visualizing the correlation between the amount of work remaining at any point in time and the progress of the project

Team(s) in reducing this work. The intersection of a trend line for work remaining and the horizontal axis indicates the most probable completion of work at that point in time. This allows me to "what if" the project by adding and removing functionality from the release to get a more acceptable date or extend the date to include more functionality. The burndown chart is the collision of reality (work done and how fast it's being done) with what is planned, or hoped for. (7)

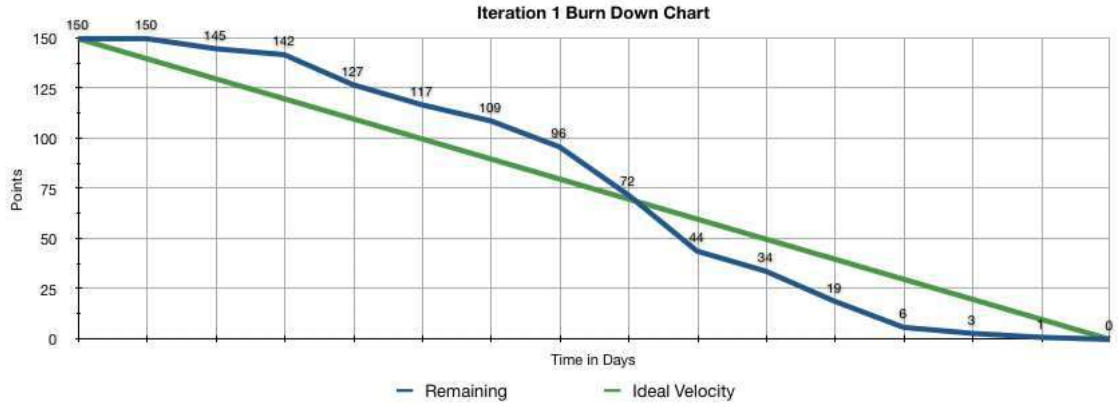

Figure 5 - Burndown Chart (6)

3.5.7 Sprint Backlog

The Sprint Backlog defines the work, or tasks, that a Team defines for turning the Product Backlog it selected for that Sprint into an increment of potentially shippable product functionality. The Team compiles an initial list of these tasks in the second part of the Sprint planning meeting. Tasks should be divided so that each takes roughly 4 to 16 hours to finish. Tasks longer than 4 to 16 hours are considered mere placeholders for tasks that haven't yet been appropriately defined. Only the Team can change the Sprint Backlog. The Sprint Backlog is a highly visible, real-time picture of the work that the Team plans to accomplish during the Sprint. An example Sprint Backlog is shown in Figure 1-6. The rows represent Sprint Backlog tasks; the columns represent the 30 days in the Sprint.

Once a task is defined, the estimated number of hours remaining to complete the task is placed in the intersection of the task and the Sprint day by the person working on the task.

				Hours of work remaining unt											
Task Description	Originator	Responsible	Status (Not Started/ In Progress/ Completed)	1	2	3	4	5	6	7	8	9	10	11	12
Meet to discuss the goals and features for Sprint 3-6	Danielle	Danielle/ Sue	Completed	20	0	0	0	0	0	0	0	0	0	0	0
Move Calculations out of Crystal Reports	Jim	Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Get KEG Data		Tom	Completed	12	0	0	0	0	0	0	0	0	0	0	0
Analyse KEG Data - Title		George	In Progress	24	24	24	24	12	10	10	10	10	10	10	10
Analyse KEG Data - Parcel		Tim	Completed	12	12	12	12	12	4	4	4	0	0	0	0
Analyse KEG Data - Encumbrance		Josh	In Progress							12	10	10	10	10	10
Analyse KEG Data - Contact		Danielle	In Progress	24	24	24	24	12	10	8	6	6	6	6	6
Analyse KEG Data - Facilities		Allen	In Progress	24	24	24	24	12	10	10	10	10	10	10	10
Define & build Database		Barry/ Dave	In Progress	80	80	80	80	80	80	60	60	60	60	60	60
Validate the size of the KEG database		Tim	Not Started												
Look at KEG Data on the G:\		Dave	In Progress	3	3	3	3	3	3	3	3	3	3	3	3
Confirm agreement with KEG		Sue	Not Started												
Confirm KEG Staff Availablity		Tom	Not Started	1	1	1	1	1	1	1	1	1	1	1	1
Switch JDK to 1.3.1. Run all tests.		Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Store PDF files in a structure		Jacquie	Completed	8	0	0	0	0	0	0	0	0	0	0	0
TopLink. Cannot get rid of netscape parser		Richard	Comleted	4	0	0	0	0	0	0	0	0	0	0	0
Buld test data repository		Barry	In Progress	10	10	10	10	10	10	10	10	8	8	8	8
Move application and database to Qual (incl Crystal)		Richard	Completed	4	4	4	4	4	4	4	0	0	0	0	0
Set up Crystal environment		Josh	Completed	2	2	2	2	1	1	1	0	0	0	0	0
Test App in Qual		Sue	In Progress												20
Defining sprint goal required for solution in 2002		Lynne	In Progress	40	40	40	40	40	40	40	38	38	38	38	38
Reference tables for import process		Josh	In Progress												
Build standard import exception process		Josh	In Progress									12	12	12	10
Handle multiple file imports on same page		Jacquie	Disregarded												
Migrate CruiseControl Servlet to iWS 6.0 (landcc_7101) server		Allen	Not Started	4	4	4	4	4	4	4	4	4	4	4	4

Figure 6 - Sprint Backlog (3)

3.6 Extreme Programming

Extreme Programming (XP) is the most widely used agile methodology. XP shares the values espoused by the Agile Manifesto for Software Development but goes further to specify a simple set of practices. Whereas many popular methodologies try to answer the question “What are all of the practices I might ever need on a software project?,” XP simply asks, “What is the simplest set of

practices I could possibly need and what do I need to do to limit my needs to those practices?" The significance of this difference cannot be understated. The most frequent critique of XP is that it is too simple to work beyond a narrow set of project criteria. Yet, the set of known successes with XP continues to stretch the breadth of projects applicable for XP. It would seem that the parameters that we use to determine what methods are appropriate for what project are still inadequate. To many, XP is a set of 12 interdependent software development practices. Used together, these practices have had much success, initially with small teams, working on projects with high degrees of change. However, the more one works with XP, the more it is apparent that the practices do not capture the essence of XP. As with the heavier methods, some teams have great success with the XP practices, some less so. Some larger teams have greater success than smaller ones. Some teams with legacy code have success; others do not. There is something more than just the practices that enables teams to succeed with XP. This extra attribute of XP is XP Values.

3.6.1 XP Introduction

Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation. In XP, every contributor to the project is a member of the "Whole Team," a single business/ development/testing team that handles all aspects of the development. Central to the team is the "Customer," one or more business representatives who sit with the team and work with them daily. XP teams use a simple form of planning and tracking to decide what to do next and to predict when any desired feature set will be delivered. Focused on business value, the team produces the software in a series of small, fully integrated releases that pass

all the tests that the Customer has defined. The core XP practices for the above are called Whole Team, Planning Game, Small Releases, and Acceptance Tests. There are specific recommendations for all of these, which are briefly discussed here and as the chapter progresses. Extreme Programmers work together in pairs and as a group, with simple design and obsessively tested code, improving the design continually to keep it always just right for the current needs. The core XP practices here are Pair Programming, Simple Design, Test-Driven Development, and Design Improvement. The XP team keeps the system integrated and running all the time. The programmers write all production code in pairs, and all work together all the time. (8)

3.6.2 XP Values

The XP Values are Communication, Simplicity, Feedback, and Courage. The essence [of XP] truly is simple. Be together with your customer and fellow programmers, and talk to each other. Use simple design and programming practices, and simple methods of planning, tracking, and reporting. Test your program and your practices, using feedback to steer the project. Working together this way gives the team courage. These values guide our actions on the project. The practices leverage these values to remove complexity from the process. The impact of the XP Values is significant and unique. XP remains the only methodology that is explicit in its values and practices. This combination gives specific guidance not only on what (the practices) to do on a project, but also on how to react (defer to the values) when the practices do not seem to be working or are not sufficient. Most methods are specific on practices, some specify principles, but few combine both. For example, CMMI describes Key Practice Areas (KPAs) but does not articulate a set of values or principles. RUP provides guiding principles, such as Develop Iteratively, but does not include values that give guidance beyond the software development practices.

Organization On a project using XP, there are two explicit roles or teams defined: the Customer and the Programmer. In keeping with the value of simplicity, most of the XP literature describes the customer as a single person who can represent the requirements, acceptance criteria, and business value for the project. In practice, it is a team of people that communicates with one voice with the Programming Team. As such, this role is also referred to as the Customer Team. This chapter uses the term "Customer" to describe the role, whether acted on by an individual or a team.The Programmer is a member of the Programming Team that implements the XP Customer Team's requirements. Again, the convention will be to use the term "Programmer" to describe an individual or the team. On all but the smallest projects, there will also be a Management Team that allocates resources for the teams, manages the alignment of the project to the goals of the business, and removes any obstacles impeding the team's progress. Extreme Programming does not specify management practices. XP attempts to simplify management by empowering the Customer and Programmer to make most of the decisions regarding the project. Often, XP teams are described as self-managing. As projects grow in size and complexity, more management is typically required to coordinate the efforts of different teams. Many of the other emerging agile methodologies are focusing more attention on management practices, such as Scrum, Lean Development and Extreme Project Management. (8)

3.6.3 The Rhythm of an XP Project

An XP project proceeds in iterations of two weeks in length. Each iteration delivers fully developed and tested software that meets the most valuable small set of the full project's requirements. Figure 4 shows the primary activities of the Customer and Programmer during the initial iterations of a project. The project

proceeds in a steady rhythm of delivering more functionality. The Customer determines at what point in time the full system can be released and deployed.

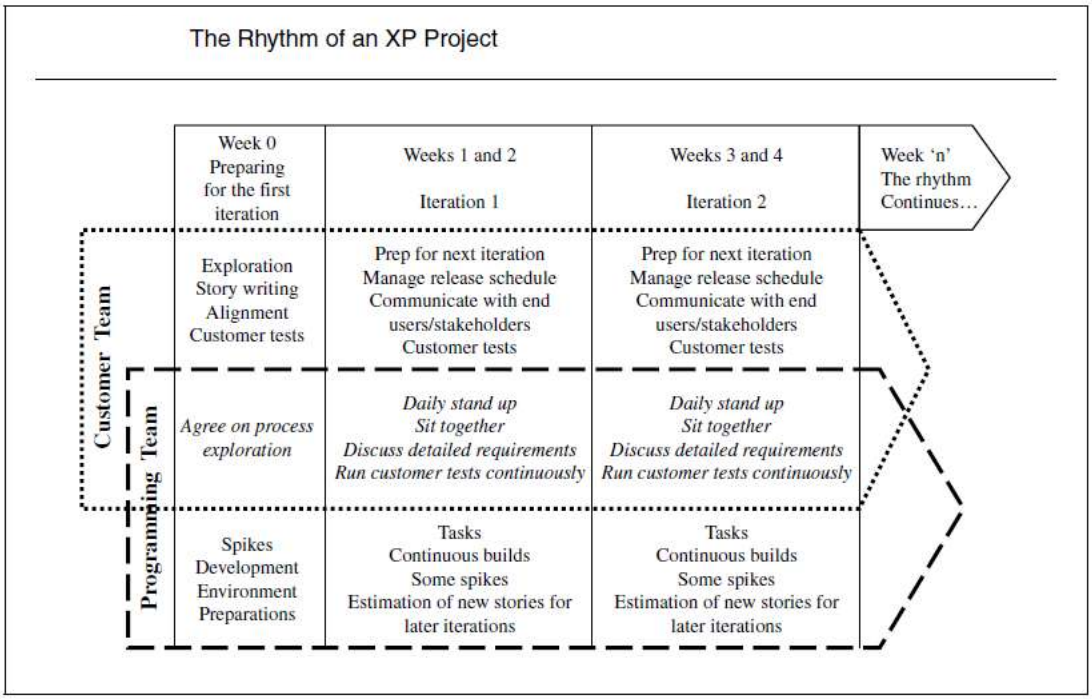

Figure 7 - The Rhythm of an XP Project (8)

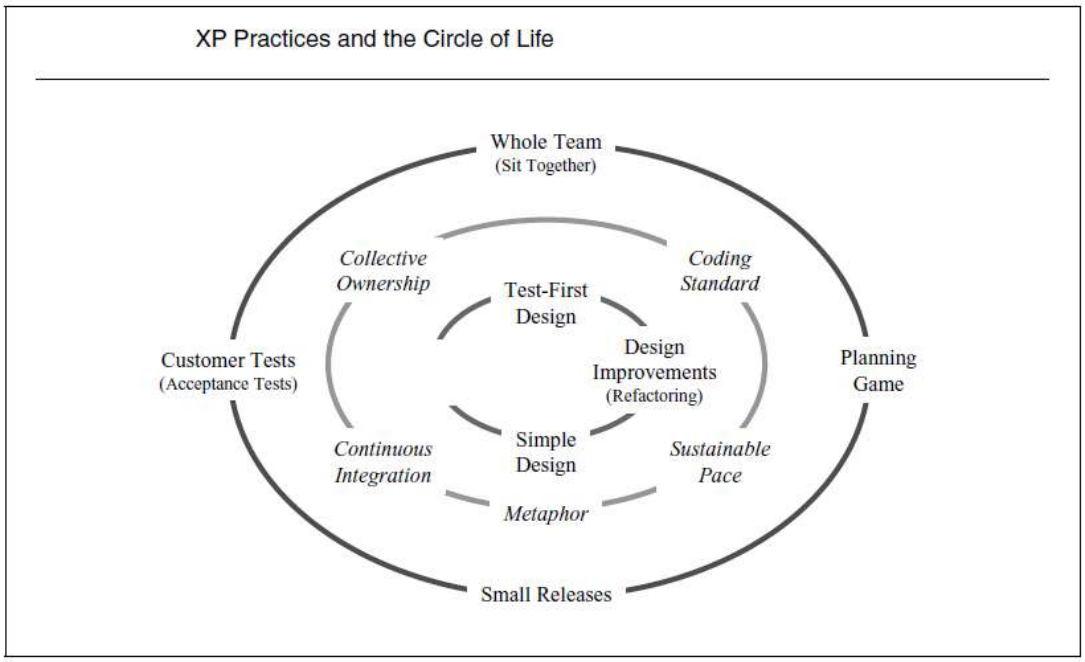

Figure 8 - XP Practices and the Circle of Life (8)

3.6.4 Core Practices

There are 12 core practices that define XP. Teams new to XP should focus on using and developing skills with these practices. Over time, as the team matures in its use of XP, it will continue to check its proficiency with these practices, but will also tailor the practices to the project needs. XP teams are encouraged to use feedback from their project to adapt, add, and eliminate practices as needed. A number of other practices are popular on XP teams and some of these are described later. The practices can be described as a cycle of activities (see Figure 5). The inner circle describes the tight cycle of the Programmers. The outer loop describes the planning cycle that occurs between the Customers and Programmers. The middle loop shows practices that help the team communicate and coordinate the delivery of quality software. (9)

3.6.5 Whole Team

All the contributors to an XP project sit together as members of one team. This team must include a business representative — the Customer — who provides the requirements, sets the priorities, and steers the project. It is best if the Customer or one of her aides is a real end user who knows the domain and what is needed. The team will, of course, have programmers. The team will typically include testers, who help the Customer define the customer acceptance tests. Analysts may serve as helpers to the Customer, helping to define the requirements. There is commonly a coach who helps the team stay on track and facilitates the process. There may be a manager, providing resources, handling external communication, and coordinating activities. None of these roles is necessarily the exclusive property of just one individual. Everyone on an XP team contributes in any way that he or she can. The best teams have no specialists, only general contributors with special skills. (8)

3.6.6 Planning Game

XP planning addresses two key questions in software development: predicting what will be accomplished by the due date, and determining what to do next. The emphasis is on steering the project — which is quite straightforward — rather than on exact prediction of what will be needed and how long it will take — which is quite difficult. There are two key planning steps in XP: 1. Release planning is a practice where the Customer presents the desired features to the programmers, and the programmers estimate their difficulty. With the cost estimates in hand, and with knowledge of the importance of the features, the Customer lays out a plan for the project. Initial release plans are necessarily imprecise; neither the priorities nor the estimates are truly solid, and until the team begins to work, we will not know just how fast they will go. Even the first

release plan is accurate enough for decision making, however, and XP teams revise the release plan regularly. 2. Iteration planning is the practice whereby the team is given direction every couple of weeks. XP teams build software in twoweek “iterations,” delivering running, useful software at the end of each iteration. During Iteration Planning, the Customer presents the features desired for the next two weeks. The programmers break them down into tasks and estimate their cost (at a finer level of detail than in Release Planning). Based on the amount of work accomplished in the previous iteration, the team signs up for what will be undertaken in the current iteration. These planning steps are very simple yet they provide very good information and excellent steering control in the hands of the Customer. Every couple of weeks, the amount of progress is entirely visible. There is no “90 percent done” in XP: a feature story was completed, or it was not. This focus on visibility results in a nice little paradox. On the one hand, with so much visibility, the Customer is in a position to cancel the project if progress is not sufficient. On the other hand, progress is so visible, and the ability to decide what will be done next is so complete, that XP projects tend to deliver more of what is needed, with less pressure and stress.

3.6.7 Customer Tests

As part of presenting each desired feature, the XP Customer defines one or more automated acceptance tests to show that the feature is working. The team builds these tests and uses them to prove to themselves, and to the customers, that the feature is implemented correctly. Automation is important because in the press of time, manual tests are skipped. That is like turning off your lights when the night gets darkest. The best XP teams treat their customer tests the same way they do programmer tests: once the test runs, the team keeps it running correctly thereafter. This means that the system only improves, always notching forward, and never backsliding. (10)

3.6.8 Small Releases

XP teams practice small releases in two important ways. First, the team releases running, tested software, delivering business value chosen by the Customer, with every iteration. The Customer can use this software for any purpose, either for evaluation or even for release to end users (which is highly recommended). The most important aspect is that the software is visible, and given to the customer at the end of every iteration. This keeps everything open and tangible. Second, XP teams also release software to their end users frequently. XP Web projects release as often as daily, inhouse projects monthly or more frequently. Even shrink-wrapped products are shipped as often as quarterly. It might seem impossible to create good versions this often but XP teams are doing it all the time. (8)

3.6.9 Simple Design

XP teams build software to a simple design. They start simple, and through programmer testing and design improvement, they keep it that way. An XP team keeps the design exactly suited for the current functionality of the system. There is no wasted motion, and the software is always ready for what is next. Design in XP is neither a one-time thing nor an up-front thing, but it is an all-the-time thing. There are design steps in release planning and iteration planning, plus teams engage in quick design sessions and design revisions through refactoring, throughout the course of the entire project. In an incremental, iterative process like Extreme Programming, good design is essential.

3.6.10 Pair Programming

In XP, two programmers, sitting side by side at the same machine, build all production software. This practice ensures that all production code is reviewed by at least one other programmer, resulting in better design, better testing, and

better code. It may seem inefficient to have two programmers doing “one programmer’s job,” but the reverse is true. Research on pair programming shows that pairing produces better code in about the same time as programmers working singly. That is right: two heads really are better than one! It does take some practice to do well, and you need to do it well for a few weeks to see the results. Most programmers who learn pair programming prefer it, so we highly recommend it to all teams. Pairing, in addition to providing better code and tests, also serves to communicate knowledge throughout the team. As pairs switch, everyone gets the benefits of everyone’s specialized knowledge. Programmers learn, their skills improve, and they become move valuable to the team and to the company. Pairing, even on its own outside of XP, is a big win for everyone. (8)

3.6.11 Test-Driven Development

XP is obsessed with feedback; and in software development, good feedback requires good testing. XP teams practice “test-driven development,” working in very short cycles of adding a test, then making it work. Almost effortlessly, teams produce code with nearly 100 percent test coverage, which is a great step forward in most shops. It is not enough to write tests; you have to run them. Here, too, XP is extreme. These “programmer tests,” or “unit tests,” are all collected together, and every time any programmer releases any code to the repository (and pairs typically release twice a day or more), every single one of the programmer tests must run correctly. One hundred percent, all the time! This means that programmers get immediate feedback on how they are doing. Additionally, these tests provide invaluable support as the software design is improved.

3.6.12 Design Improvement

XP focuses on delivering business value in every iteration. To accomplish this over the course of the whole project, the software must be well designed. The alternative would be to slow down and ultimately get stuck. So, XP uses a process of continuous design improvement called "refactoring." The refactoring process focuses on the removal of duplication (a sure sign of poor design), and on increasing the "cohesion" of the code while lowering the "coupling." High cohesion and low coupling have been recognized as the hallmarks of well-designed code for at least 30 years. The result is that XP teams start with a good, simple design, and always have a good, simple design for the software. This lets them sustain their development speed and, in fact, generally increase speed as the project goes forward. Refactoring is, of course, strongly supported by comprehensive testing that ensures that as the design evolves, nothing is broken. Thus, the customer tests and programmer tests are a critical enabling factor. The XP practices support each other: they are stronger together than separately.

3.6.13 Continuous Integration

XP teams keep the system fully integrated at all times. We say that daily builds are for wimps; XP teams build multiple times per day. (One XP team of 40 people builds at least eight or ten times per day!) The benefit of this practice can be seen by thinking back on projects you may have heard about, where the build process was weekly or less frequently and usually led to "integration hell," where everything broke and no one knew why. Infrequent integration leads to serious problems on a software project. First of all, although integration is critical to shipping good working code, the team is not practiced at it, and often it is delegated to people who are not familiar with the whole system. Second, infrequently integrated code is often or usually buggy code. Problems creep in at

integration time that are not detected by any of the testing that takes place on a nonintegrated system. Third a weak integration process leads to long code freezes. Code freezes mean that you have long time periods when the programmers could be working on important shippable features, but that those features must be held back.This weakens your position in the market or with your end users. (8)

3.6.14 Collective Code Ownership

On an XP project, any pair of programmers can improve any code at anytime. This means that all code gets the benefit of many people's attention, which increases code quality and reduces defects. There is another important benefit as well: when code is owned by individuals, required features are often put in the wrong place as one programmer discovers that he needs a feature somewhere in code that he does not own.The owner is too busy to do it, so the programmer puts the feature in his own code, where it does not belong. This leads to ugly, hard-to-maintain code, full of duplication and with low (bad) cohesion. Collective ownership could be a problem if people worked blindly on code they do not understand. XP avoids these problems through two key techniques: (1) the programmer tests catch mistakes, and (2) pair programming, which means that the best way to work on unfamiliar code is to pair with the expert. In addition to ensuring good modifications when needed, this practice spreads knowledge throughout the team.

3.6.15 Coding Standard

XP teams follow a common coding standard so that all the code in the system looks as if a single — very competent — individual wrote it. The specifics of the standard are not important; what is important is that all the code looks familiar, in support of collective ownership.

3.6.16 Metaphor

XP teams develop a common vision of how the program works, which we call the “metaphor.” At its best, the metaphor is a simple, evocative description of how the program works, such as “this program works like a hive of bees, going out for pollen and bringing it back to the hive” as a description for an agent-based information retrieval system. Sometimes, a sufficiently poetic metaphor does not arise. In any case, with or without vivid imagery, XP teams use a common system of names to be sure that everyone understands how the system works and where to look to find functionality or to find the right place to put the functionality that is about to be added. (11)

3.7 Other Common Practices of XP

The core practices of XP do not specify all of the activities that are required to deliver a software project. As teams use XP, many find that other practices aid in their success, in some cases as significantly as some of the core practices. The following are some other practices commonly used by successful XP teams.

3.7.1 Open Workspace

To maximize communication among the Whole Team, the team works together in an“open workspace.” This is a large room, with tables in the center that can typically seat two to four pairs of developers. By sitting together, all team members can establish instant communication when needed for the project. Teams establish their own rules concerning their space to ensure that everyone can work effectively. The walls of the “open workspace” are used to display information about the project. This will include big, visible charts of metrics such as passing acceptance tests and team productivity. There may be designs drawn

on whiteboards. Project status will be displayed so that any participant or stakeholder of the project can always see progress. (8)

3.7.2 Retrospectives

The XP practices provide feedback to the team as to the quality of the code and its alignment to the Customers' needs. The team also needs feedback on how it is performing. Is it following the practices with discipline? Are there adaptations to the practices that would benefit the team? The practice commonly used for this is the Retrospective. After each iteration, the team does a short reflection on what went well during the iteration and what should be improved in the next iteration. After a release of the product, a more in-depth Retrospective is performed on the whole project.

3.7.3 Self-Directed Teams

A practice that is common among most of the agile methods is self-directed teams. The best people to make decisions about the project are those closest to the details, as long as they have an understanding of the overall goals of the project. Open communication allows team members to have the information required to make decisions. Managers are part of the communication loop but not bottlenecks in the decision-making flow.

3.7.4 Customer Team

As XP is used on projects with more complex requirements, a team performs the Customer function. For larger or more complex projects, the Customer team may even exceed the Programming team in size. Some of the challenges faced by the Customer team include communicating with and balancing the needs of multiple stakeholders, allocating resources to the appropriate projects or features, and providing sufficient feedback to ensure that the requirements implemented

achieve the stakeholders' goals. The specific Customer team practices are still emerging in the agile community. The practices are guided by the same values as the other XP practices. (8)

Probably the most commonly debated question regarding XP is whether it can be used successfully on a particular type of project. Experience is proving that, as with other approaches to software development, the limitations often include the characteristics of the project, the people on the team, and the organization in which they work. To evaluate whether the XP practices can help a team achieve greater success on their project, consideration must be given to the project characteristics, the people on the team, and the cultures of the organizations involved in the project. The XP Values can be used as a template to test the fit of XP to a project, team, and organization. Simply evaluate the degree to which each value is currently held by the team and the organization.

Communication. Does the team communicate constantly and effectively? Does this communication extend to the customer? Is the team's software readable and understandable (i.e., is it easy for Programmers to communicate with the code)? Simplicity. Is the team comfortable with simple solutions? Can the team implement, without a complete design, the system prior to coding? Is the team comfortable with some ambiguity as to the exact requirements and designs? Can the team adapt often to changing requirements? Is the team working new code or code that is well designed and refactored? Feedback. Can the team get feedback on its tasks and deliverables often? Does the team accept feedback constructively? When there are problems, does the team focus on the process to identify root causes (rather than the people)? How often does the team integrate, build, and test the complete software system? Courage. Does the organization

encourage individuals to not fear failure? Are individuals and teams encouraged to show initiative and make decisions for their projects? Are organizational boundaries easily crossed to solve problems on the project? Typically, the greater the degree to which the team can answer these questions affirmatively, the fewer changes will be required and the easier it is for the team and organization to adopt XP. Some Specific project and team guidelines for getting started are provided next.

3.8 Getting Started with XP

When selecting an initial project on which to try XP, one must consider the challenges of using the new practices. New practices introduce risk to a project. Care must be taken to select an initial project that is not burdened by all of the most difficult obstacles to using XP, but does address enough typical obstacles so that the success of the initial project can provide the basis for expanding to the rest of the organization. Although most initial XP projects are not this fortunate, ideally, the initial project will have many of the following characteristics:

- Primarily new code versus legacy updates
- An identified and available source of requirements and feedback
- Delivers important business value and has management visibility
- Uses an OO language/environment
- Is typical of the projects the organization will be doing in the future
- Has a co-located team in an open workspace
- Can be delivered to the end user incrementally with a new stage once in at least every four to six weeks

In selecting the initial XP Project Team, the main attribute of the team members should be a strong commitment to delivering the project and achieving its goals using the new practices. Some healthy skepticism about XP is acceptable as long as the team members are willing to use the practices and let data and experience from the project guide any adaptations. The team ideally will have a few technical leaders familiar with other projects in the organization, but it is not desirable to have a team full of the most senior people. XP is a collaborative approach to development and, as such, the initial project will benefit from members with strong "soft" skills who prefer collaborative work environments. Beyond these characteristics, the team should be representative of teams that the organization will use in the future. The simplest way to reduce risk on an initial project is maximize the skill of the team as quickly as possible. This can be achieved through recruiting team members that are already skilled in XP, training, or experienced coaching for an inexperienced team. (8)

3.8.1 Adaptations

As teams begin adopting the XP practices, numerous obstacles and constraints must be confronted. The team may have trouble gaining access to the Customer every day. The team may have trouble co-locating to an open workspace. The team may be so large that communicating without formal documentation is not feasible. How do we adjust? Must we abandon XP? The XP Values guide teams in solving these process problems with their projects. The Courage value guides us to aggressively confront and remove any obstacles that would add steps, artifacts, or complexity to the process. This often means letting common sense outweigh bureaucracy. For example, teams sometimes do not feel empowered to change the physical work environment to have an open workspace (i.e., change the cubicles). Often, a little courage, negotiating, and a power screwdriver will remove this obstacle. Some teams struggle to have a customer sitting with the

team. The programmers develop from a requirements document and have never spoken to the customer. Although the thought of having a customer present is desirable, the logistics can seem impossible, particularly if the best person to sit with the team does not live near the team or is constantly traveling. Often, with a slight reorganization and a modified communication infrastructure, a customer can be identified who can sit with the team on a frequent basis. Of course, courage can only take us so far. There will be constraints that interfere with our ability to implement the practices as described. A common example is legacy code. Many teams work with large code bases that do not have tests and are in dire need of design improvement. We want to aggressively move to the state where all of the code has passing tests, is understandable, and is well designed. The initial attempt is to rapidly get the code up to our new standard. Can we toss it and rewrite it? Would it really be that expensive and timeconsuming to fix it? Is there other, cleaner code available with which we can replace it? Very often, the answers are No, Yes, and No, respectively, leaving team members no choice but to live with the smelly code and improve as they can. XP Values give the team a helpful, simple tool to deal with this difficult, yet inevitable challenge. The constraint that causes a practice to be modified or abandoned is reviewed against each of the XP Values, using the following question: How will the influence of this XP Value be diminished as a result? In the case of our untestable legacy code, a quick brainstorming session by the team might yield the ideas in the Impact column of Table 3. The team discusses ways to adapt the process that is guided by the values, yielding something similar to the Adaptation Alternatives column. Each alternative that the team considers is checked for its alignment to the values.A misaligned example, an alternative that states “all legacy code changes must be approved by a Change Control Board (CCB) prior to implementation,” may be viable, but it is not simple to implement. It reduces

the frequency of feedback while we wait for the CCB to meet, and takes empowerment away from the programmers, thus reducing their Courage. Other alternatives that address the constraint and that align closer to the XP Values are preferred. Using this simple technique, teams adapt the XP Practices to their project and team needs. The importance of starting with Courage cannot be overstated. Many teams have been able to achieve a level of simplicity in their practices beyond what was thought possible. Although this may appear to introduce risk, Retrospectives after each iteration mitigate that risk by helping the team understand where additional adaptations are required. (8)

4 Survey

A survey study was conducted among target population of professionals with experience with agile project management. This study employed web survey to gather data.

4.1 Methodology

The methodology is based on by A survey study of critical success factors in agile software projects. (12) Basic statistical methods such as frenquencies and crosstab statistics are used to come up with a conclusion. According to the fact that there was only 32 valid responses, the Fisher´s Exact test is used to test dependency on selected variables. Moreover, the Cramer´s V is used to determine the strength of dependency.

4.1.1 Fisher´s Exact Test

Fisher's exact test is a statistical test used to determine if there are nonrandom associations between two categorical variables.

Let there exist two such variables X and Y, with m and n observed states, respectively. Now form an $m \times n$ matrix in which the entries $a_{i\,j}$ represent the number of observations in which $x = i$ and $y = j$. Calculate the row and column sums R_i and C_j, respectively, and the total sum

$$N = \sum_i R_i = \sum_j C_j \tag{1}$$

of the matrix. Then calculate the conditional probability of getting the actual matrix given the particular row and column sums, given by

$$P_{\text{cutoff}} = \frac{(R_1!\,R_2!\cdots R_m!)\,(C_1!\,C_2!\cdots C_n!)}{N!\,\prod_{i,j} a_{i\,j}!}, \tag{2}$$

which is a multivariate generalization of the hypergeometric probability function. Now find all possible matrices of nonnegative integers consistent with the row and column sums R_i and C_j. For each one, calculate the associated conditional probability using (2), where the sum of these probabilities must be 1.

To compute the P-value of the test, the tables must then be ordered by some criterion that measures dependence, and those tables that represent equal or greater deviation from independence than the observed table are the ones whose probabilities are added together. There are a variety of criteria that can be used to measure dependence. In the 2×2 case, which is the one Fisher looked at when he developed the exact test, either the Pearson chi-square or the difference in proportions (which are equivalent) is typically used. Other measures of association, such as the likelihood-ratio-test, G-squared, or any of the other measures typically used for association in contingency tables, can also be used.

The test is most commonly applied to 2×2 matrices, and is computationally unwieldy for large m or n. For tables larger than 2×2, the difference in proportion can no longer be used, but the other measures mentioned above remain applicable (and in practice, the Pearson statistic is most often used to order the tables). In the case of the 2×2 matrix, the P-value of the test can be simply computed by the sum of all P-values which are $\leq P_{\text{cutoff}}$.

For an example application of the 2×2 test, let X be a journal, say either Mathematics Magazine or Science, and let Y be the number of articles on the topics of mathematics and biology appearing in a given issue of one of these journals. If Mathematics Magazine has five articles on math and one on biology, andScience has none on math and four on biology, then the relevant matrix would be

	Math. Mag.	Science	
math	5	0	$R_1 = 5$
biology	1	4	$R_2 = 5$
	$C_1 = 6$	$C_2 = 4$	$N = 10.$

(3)

Computing P_{cutoff} gives

$$P_{\text{cutoff}} = \frac{5!^2\, 6!\quad 4!}{10!\,(5!\quad 0!\quad 1!\quad 4!)} = 0.0238, \tag{4}$$

and the other possible matrices and their Ps are

$$\begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix} P \quad = \quad 0.2381 \tag{5}$$

$$\begin{bmatrix} 3 & 2 \\ 3 & 2 \end{bmatrix} P \quad = \quad 0.4762 \tag{6}$$

$$\begin{bmatrix} 2 & 3 \\ 4 & 1 \end{bmatrix} P \quad = \quad 0.2381 \tag{7}$$

$$\begin{bmatrix} 1 & 4 \\ 5 & 0 \end{bmatrix} P \quad = \quad 0.0238, \tag{8}$$

which indeed sum to 1, as required. The sum of P-values less than or equal to $P_{\text{cutoff}} = 0.0238$ is then 0.0476 which, because it is less than 0.05, issignificant. Therefore, in this case, there would be a statistically significant association between the journal and type of article appearing. (13)

4.1.2 Cramer´s V Test

Cramer's V is a statistic measuring the strength of association or dependency between two (nominal) categorical variables in a contingency table.

Setup. Suppose X and Y are two categorical variables that are to be analyzed in a some experimental or observational data with the following information:

X has M distinct categories or classes, labeled $X_1, \ldots, X_M$,

Y has N distinct categories, labeled $Y_1, \ldots, Y_N$,

n pairs of observations (x_k, y_k) are taken, where x_i belongs to one of the M categories in X and y_i belongs to one of the N categories in Y.

Form a $M \times N$ contingency table such that Cell (i, j) contains the count n_{ij} of occurrences of Category X_i in X and Category Y_j in Y:

$X \backslash Y$	Y_1	Y_2	$\ldots$	Y_N
X_1	n_{11}	n_{12}	$\ldots$	n_{1N}
X_2	n_{21}	n_{22}	$\ldots$	n_{2N}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
X_M	n_{M1}	n_{M2}	$\ldots$	n_{MN}

Note that $n = \sum n_{ij}$.

Definition. Suppose that the null hypothesis is that X and Y are independent random variables. Based on the table and the null hypothesis, the chi-squared statistic χ^2 can be computed. Then, Cramer's V is defined to be

$$V = V(X, Y) = \sqrt{\frac{\chi^2}{n \min(M-1, N-1)}}.$$

Of course, in order for V to make sense, each categorical variable must have at least 2 categories.

Remarks.

$0 \leq V \leq 1$. The closer V is to 0, the smaller the association between the categorical variables X and Y. On the other hand, V being close to 1 is an indication of a strong association between X and Y. If $X = Y$, then $V(X, Y) = 1$.

When comparing more than two categorical variables, it is customary to set up a square matrix, where cell (i, j) represents the Cramer's V between the i th variable and the j th variable. If there are n variables, there are $\frac{n(n-1)}{2}$ Cramer's V's to calculate, since, for any discrete random variables X and Y, $V(X, X) = 1$ and $V(X, Y) = V(Y, X)$. Consequently, this matrix is symmetric. If one of the categorical variables is dichotomous, (either M or $N = 2$), Cramer's V is equal to the phi statistic (Φ), which is defined to be

$$\Phi = \sqrt{\frac{\chi^2}{n}}.$$

Cramer's V is named after the Swedish mathematician and statistician Harald Cramér, who sought to make statistics mathematically rigorous, much like Kolmogorov's axiomatization of probability theory. Cramér also made

contributions to number theory, probability theory, and actuarial mathematics widely used by the insurance industry. (14)

4.2 Data collection

A web survey with Likert scale questionnaires and demographic information collection was distributed to the target population. There were three sections in the survey.

The first section was on demographic data, which included both the respondent's demographic information as well as the agile project information.

The second section was on success factors. To measure importance of success factors, a 5-point Likert scale was used to reflect the level of perception of the question by the respondent.

The third section was on perception of success, and again, to measure perception of success of agile projects, a 5-point Likert scale was used to reflect the level of perception of the question by the respondent.

4.3 General Variables

The survey asked for four general variables:

- Method of the project management used
    - Extreme Programming
    - SCRUM
    - FDD
    - Other
- Project Location
    - West Europe
    - Central Europe

    - East Europe
- Number of team members in a project
    - <3
    - 3<6
    - 6<12
    - 12<50
    - More
- Duration of a project
    - 1<3
    - 3<6
    - 6<12
    - 12<24
    - more

4.4 Describtive statistics of the survey

There is describtive statistics of the survey below. Frequency tables and pie charts were computed.

4.4.1 Method used

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	XP	12	37,5	37,5	37,5
	Scrum	8	25,0	25,0	62,5
	FDD	6	18,8	18,8	81,3
	Other	6	18,8	18,8	100,0
	Total	32	100,0	100,0	

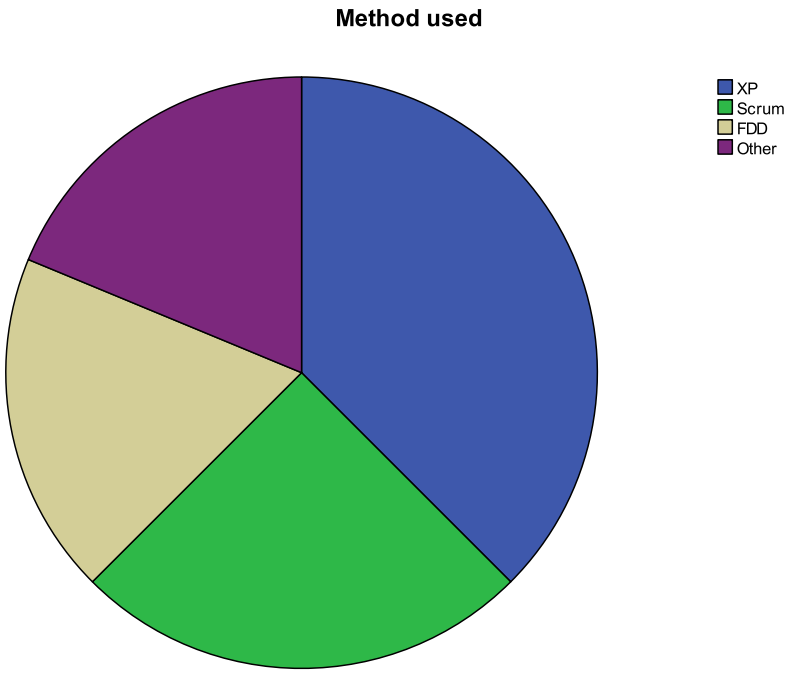

Figure 9 - Method Used

As seen XP programming and Scrum used more than half of the respondes of this survey.

4.4.2 Size of a team

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	<3	12	37,5	37,5	37,5
	3<6	18	56,3	56,3	93,8
	12<50	2	6,3	6,3	100,0
	Total	32	100,0	100,0	

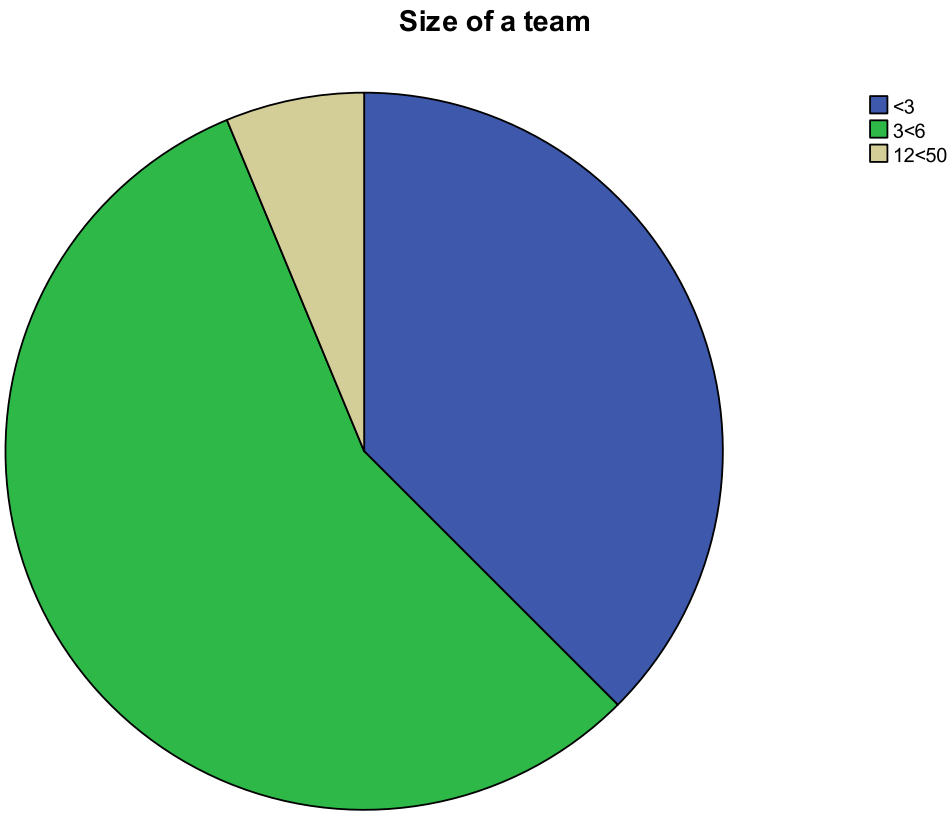

Figure 10 - Size of a Team

The team of the projects consisted almost in all cases of up to 6 people. According to this the survey is considering small projects mainly.

4.4.3 Location

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	West EU	4	12,5	12,5	12,5
	CEE	27	84,4	84,4	96,9
	EEU	1	3,1	3,1	100,0
	Total	32	100,0	100,0	

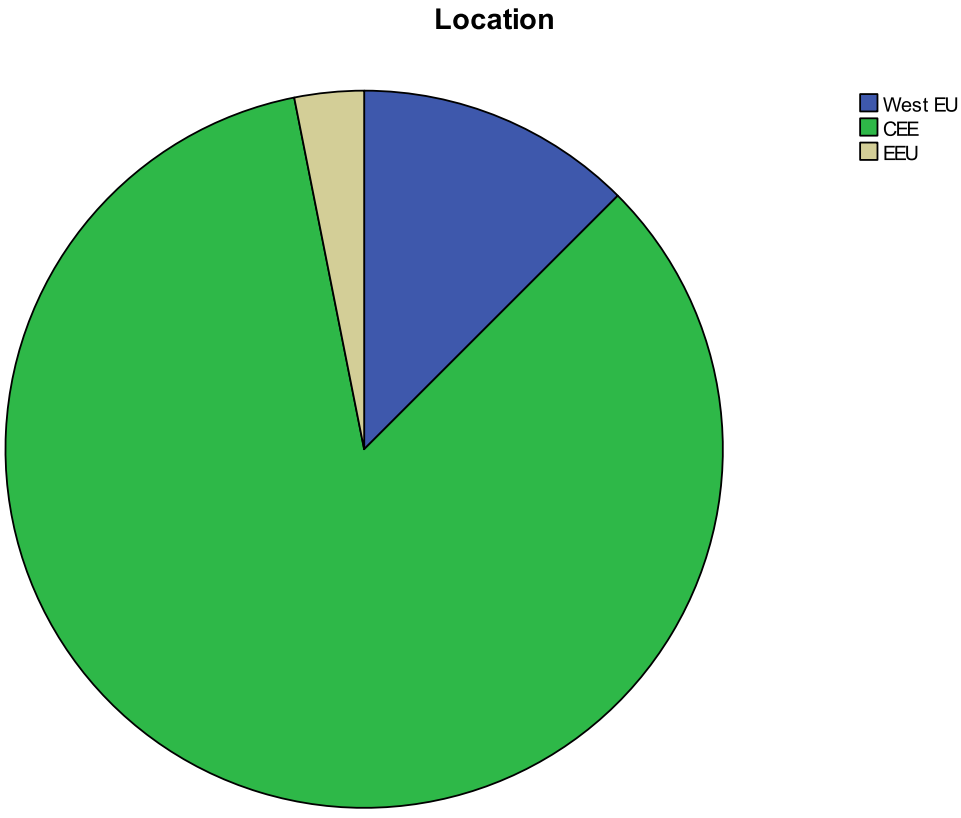

Figure 11 - Location

The projects were held mainly in Cental and Eastern Europe.

4.4.4 Size in months

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1<3	3	9,4	9,4	9,4
	3<6	12	37,5	37,5	46,9
	6<12	11	34,4	34,4	81,3
	12<24	4	12,5	12,5	93,8
	<24	2	6,3	6,3	100,0
	Total	32	100,0	100,0	

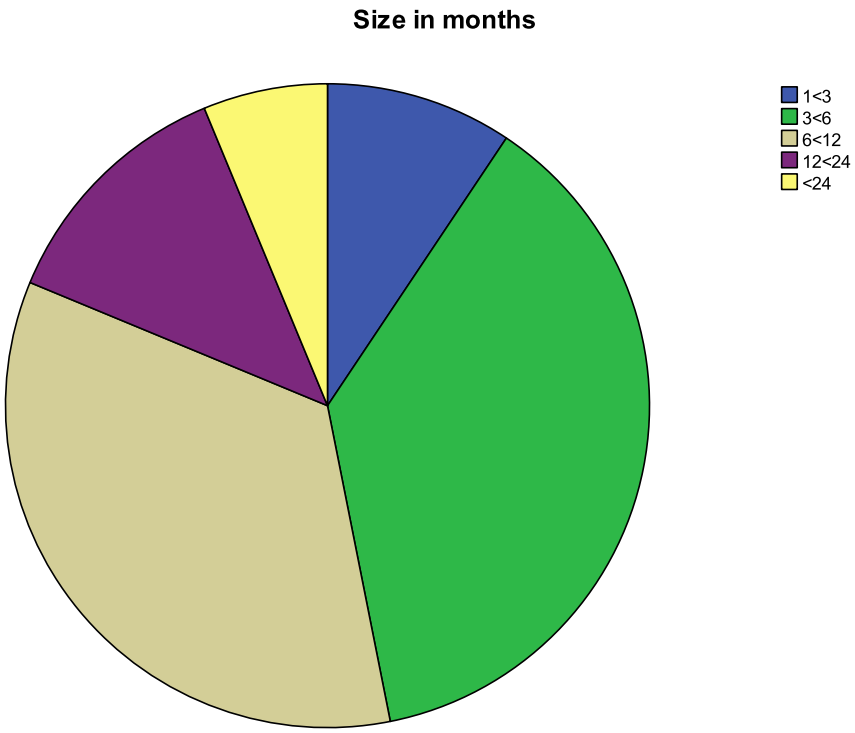

Figure 12 - Size in Months

Majority of the projects have duration below one year. This can be seen as consistent according to the number of people involven see Size of a Team chart.

4.5 Success Atributes

According to the methodology used in A survey study of critical success factors in agile software projects (12) there are four success attributes. All of them shows the success perception of the responder in particular dimension by 5-point Likert scale.

1. Quality (delivering good product or project outcome)

2. Scope (meeting all requirements andobjectives)

3. Time (delivering on time)

4. Cost (delivering within estimated cost and effort)

4.5.1 Quality

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	9	28,1	28,1	28,1
	3	19	59,4	59,4	87,5
	4	4	12,5	12,5	100,0
	Total	32	100,0	100,0	

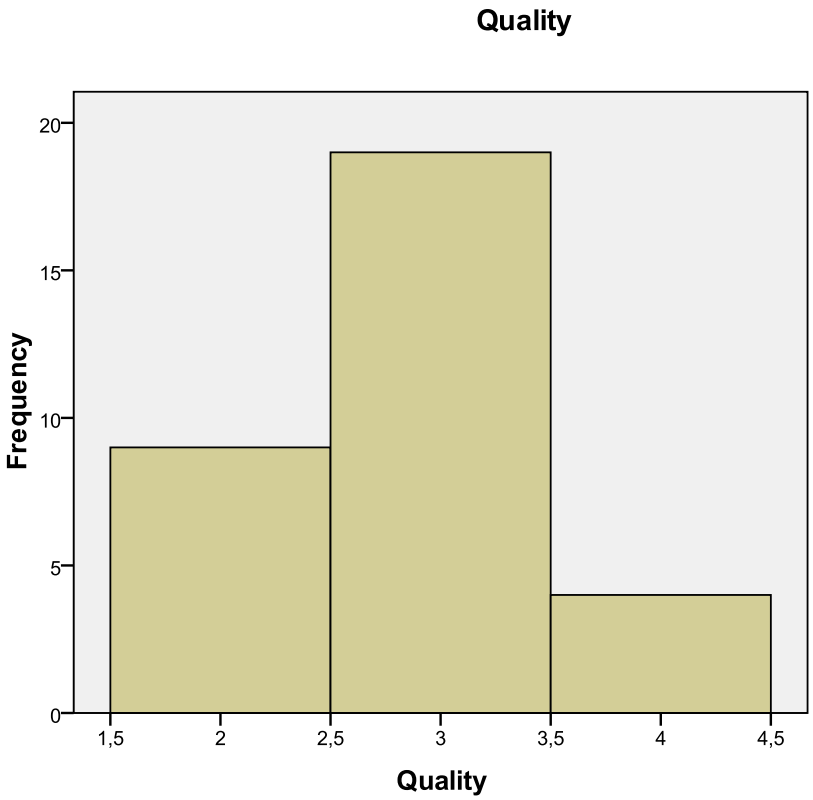

Figure 13 - Quality

According to the level of quality of the final product of the project, the quality was considered as average and below average in the survey.

4.5.2 Time

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	4	12,5	12,5	12,5
	2	14	43,8	43,8	56,3
	3	9	28,1	28,1	84,4
	4	5	15,6	15,6	100,0
	Total	32	100,0	100,0	

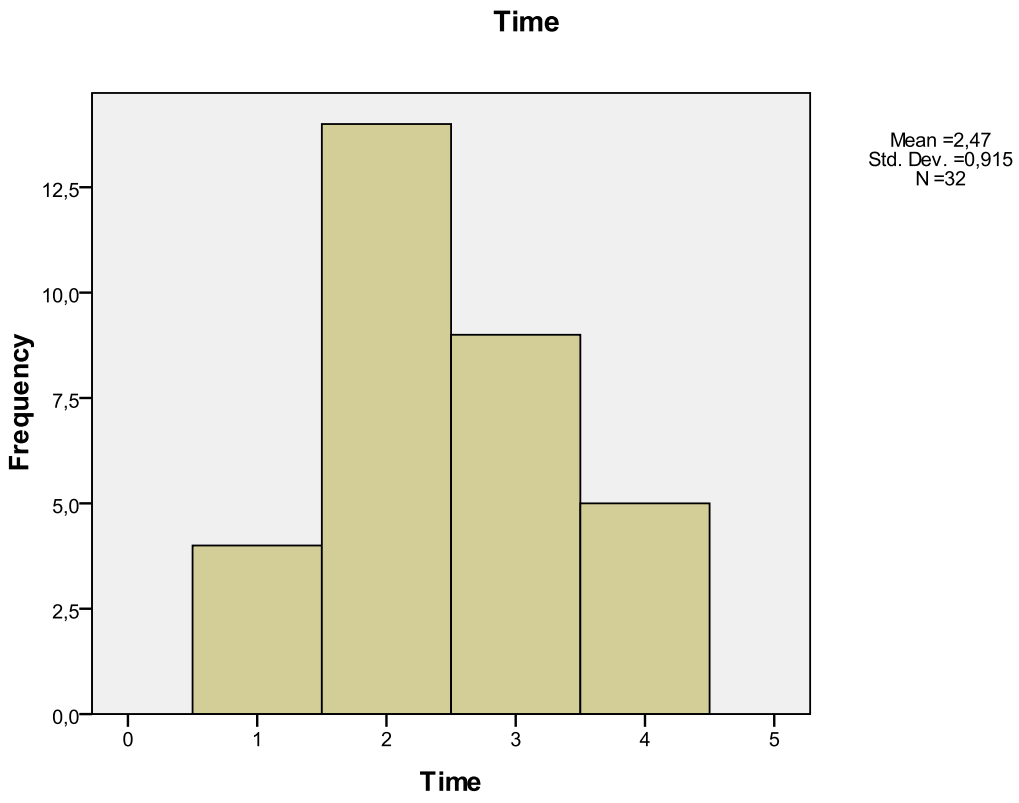

Figure 14 - Time

Only approximately a quarter of all projects met the criterium time above the level of average. Most of the projects have a problem of delivering a product on time.

4.5.3 Scope

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	4	12,5	12,5	12,5
	3	19	59,4	59,4	71,9
	4	9	28,1	28,1	100,0
	Total	32	100,0	100,0	

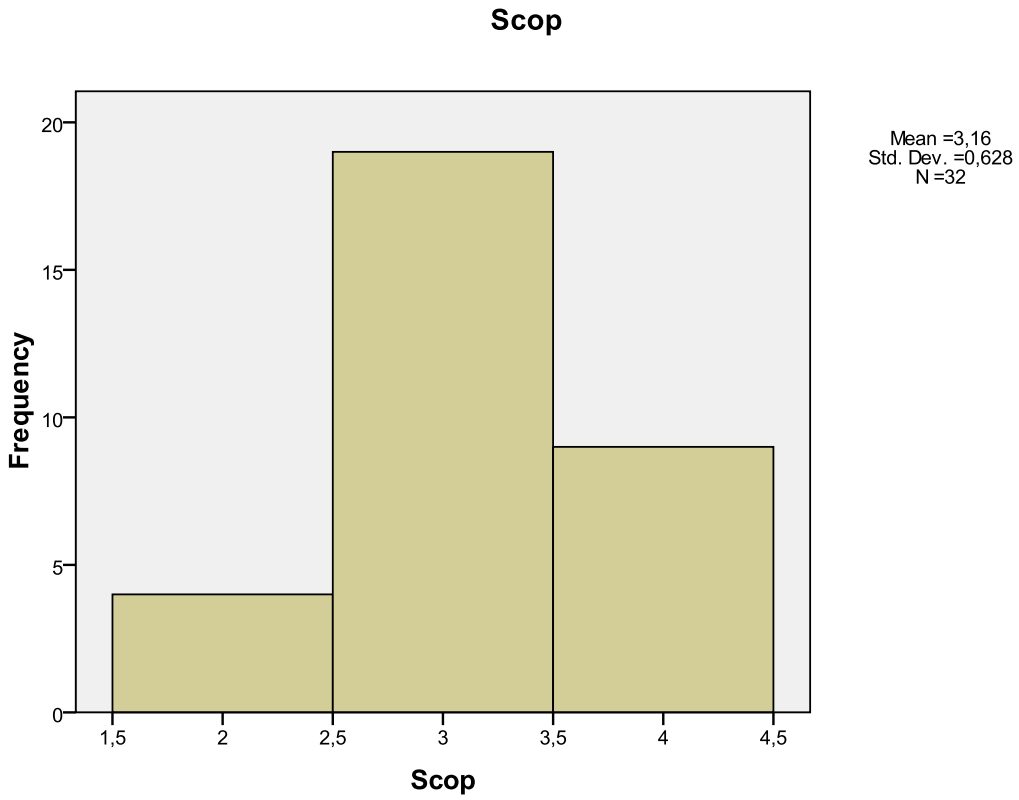

Figure 15 - Scope

A scope of the project was considered successful as average and above the average by the respondents.

4.5.4 Cost

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	3	5	15,6	15,6	15,6
	4	13	40,6	40,6	56,3
	5	14	43,8	43,8	100,0
	Total	32	100,0	100,0	

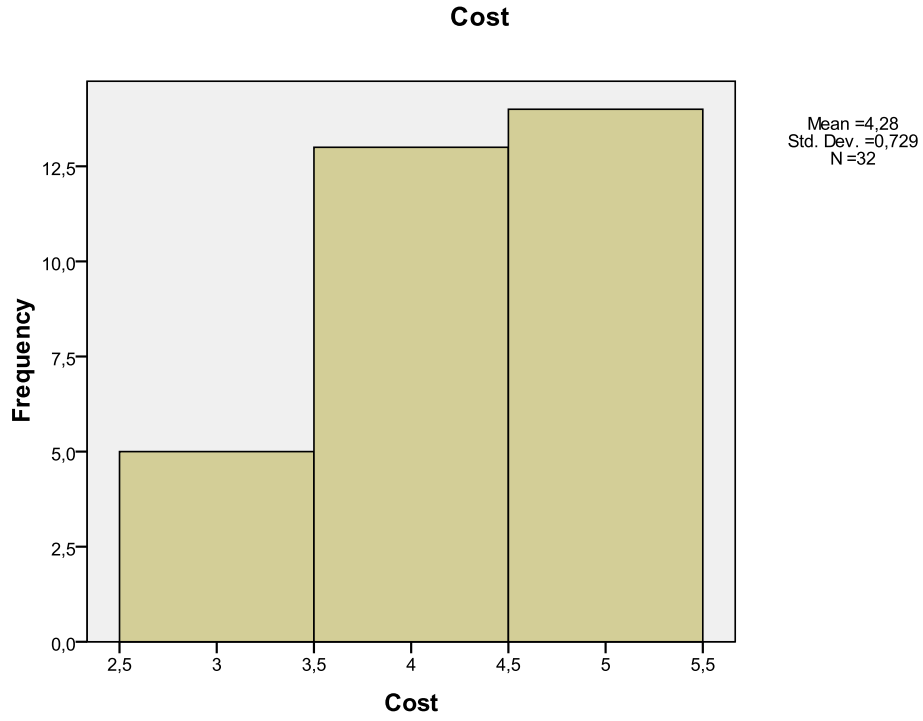

Figure 16 – Cost

The criterion of Cost was met significantly more than any other in a survey, almost 50 % of respondents stated that they met the cost of their projects.

4.6 Crosstab analysis

The contingency tables were computed in SPSS version 17 in order to see dependencies of basic evident information of the projects and success perception factors. The aim is to try to find some statisticly significant evidence whether for instance method used in a project has an impact on the success.

4.6.1 Method used * Quality

H_0: Method that was used in a project is independent on the success perception of the Quality of the project.

H_A: Quality of the project depends on the method used during the project.

The estimation is that agile method that is chosen will have an influence on the quality of a project.

Crosstab

Count

		Quality			
		2	3	4	Total
Method used	XP	2	8	2	12
	Scrum	0	6	2	8
	FDD	3	3	0	6
	Other	4	2	0	6
Total		9	19	4	32

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)
Pearson Chi-Square	10,877[a]	6	,092	,087
Likelihood Ratio	13,503	6	,036	,065
Fisher's Exact Test	9,628			,080
N of Valid Cases	32			

a. 11 cells (91,7%) have expected count less than 5. The minimum expected count is ,75.

Symmetric Measures

		Value	Approx. Sig.	Exact Sig.
Nominal by Nominal	Phi	,583	,092	,087
	Cramer's V	,412	,092	,087
N of Valid Cases		32		

Conclusion: Null hypothesis is not possible to reject on the significance level $\alpha = 0.05$. According to Fisher´s Exact test both variables are independend.

Result: Method that was used in a project is independent on the success perception of the Quality of the project.

4.6.2 Method used * Time

H_0: Method that was used in a project is independent on the success perception of delivering the project on time.
H_A: Delivering project on time depends on the method used during the project.

The estimation is that agile method that is chosen will have an influence on the delivery time of the project.

Crosstab

Count

		Time				
		1	2	3	4	Total
Method used	XP	2	2	7	1	12
	Scrum	0	2	2	4	8
	FDD	0	6	0	0	6
	Other	2	4	0	0	6
Total		4	14	9	5	32

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)
Pearson Chi-Square	27,344[a]	9	,001	,001
Likelihood Ratio	30,055	9	,000	,001
Fisher's Exact Test	21,056			,001
N of Valid Cases	32			

a. 15 cells (93,8%) have expected count less than 5. The minimum expected count is ,75.

Symmetric Measures

		Value	Approx. Sig.	Exact Sig.
Nominal by Nominal	Phi	,924	,001	,001

	Cramer's V	,534	,001	,001
N of Valid Cases		32		

Conclusion: Null hypothesis is rejected on significance level 0.05. According to Fisher´s Exact test both variables are dependend. Cramer´s V statistic shows strong strength of dependence between Method and timeframe of a project.

4.6.3 Method Used * Scope

H_0: Method that was used in a project is independent on meeting scope of the project.
H_A: Meeting scope of the project depends on the method used during the project.

The estimation is that agile method that is chosen will have an influence on the meeting scope of the project.

Crosstab

Count

		Scop			
		2	3	4	Total
Method used	XP	2	8	2	12
	Scrum	2	6	0	8
	FDD	0	3	3	6
	Other	0	2	4	6
Total		4	19	9	32

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)
Pearson Chi-Square	10,877[a]	6	,092	,087
Likelihood Ratio	13,503	6	,036	,065

Fisher's Exact Test	9,628			,080
N of Valid Cases	32			

a. 11 cells (91,7%) have expected count less than 5. The minimum expected count is ,75.

Symmetric Measures

		Value	Approx. Sig.	Exact Sig.
Nominal by Nominal	Phi	,583	,092	,087
	Cramer's V	,412	,092	,087
N of Valid Cases		32		

Conclusion: Null hypothesis is not possible to reject on the significance level $\alpha = 0.05$. According to Fisher´s Exact test both variables are independend.

4.6.4 Method used * Cost

H_0: Method that was used in a project is independent on the success perception of meeting cost of the project.

H_A: Cost of the project depends on the method used during the project.

The estimation is that agile method that is chosen will have an influence on the meeting cost of the project.

Crosstab

Count

		Cost			
		3	4	5	Total
Method used	XP	0	4	8	12
	Scrum	0	2	6	8

	FDD	3	3	0	6
	Other	2	4	0	6
Total		5	13	14	32

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)
Pearson Chi-Square	19,112[a]	6	,004	,003
Likelihood Ratio	24,901	6	,000	,001
Fisher's Exact Test	17,933			,001
N of Valid Cases	32			

a. 11 cells (91,7%) have expected count less than 5. The minimum expected count is ,94.

Symmetric Measures

		Value	Approx. Sig.	Exact Sig.
Nominal by Nominal	Phi	,773	,004	,003
	Cramer's V	,546	,004	,003
N of Valid Cases		32		

Conclusion: Null hypothesis is rejected on significance level 0.05. According to Fisher´s Exact test both variables are dependend. Cramer´s V statistic shows rather strong strength of dependence between Method and meeting cost of a project.

4.6.5 Size of a team * Quality

H_0: Size of a team that was used in a project is independent on the success perception of Quality of the project.

H_A: Quality of the project depends on the size of the team working on the project.

The estimation is that size of a team that is chosen will have an influence on the quality of a project.

Crosstab

Count

		Quality			
		2	3	4	Total
Size of a team	<3	2	7	3	12
	3<6	6	11	1	18
	12<50	1	1	0	2
Total		9	19	4	32

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)
Pearson Chi-Square	3,559[a]	4	,469	,514
Likelihood Ratio	3,676	4	,452	,554
Fisher's Exact Test	3,690			,498
N of Valid Cases	32			

a. 6 cells (66,7%) have expected count less than 5. The minimum expected count is ,25.

Symmetric Measures

	Value	Approx. Sig.	Exact Sig.

Nominal by Nominal	Phi	,334	,469	,514
	Cramer's V	,236	,469	,514
N of Valid Cases		32		

Conclusion: It is not possible to reject null hypothesis on the significance level 0.05. According to Fisher´s Exact test both variables are independend.

4.6.6 Size of a team * Time

H_0: Size of a team that was used in a project is independent on the success perception of meeting timeframe of the project.
H_A: Timeframe of the project depends on the size of the team working on the project.
The estimation is that size of a team that is chosen will have an influence on delivering project ont time.

Crosstab

Count

		Time				
		1	2	3	4	Total
Size of a team	<3	0	6	5	1	12
	3<6	3	8	3	4	18
	12<50	1	0	1	0	2
Total		4	14	9	5	32

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)
Pearson Chi-Square	8,169[a]	6	,226	,209

Likelihood Ratio	9,855	6	,131	,174
Fisher's Exact Test	7,739			,208
N of Valid Cases	32			

a. 9 cells (75,0%) have expected count less than 5. The minimum expected count is ,25.

Symmetric Measures

		Value	Approx. Sig.	Exact Sig.
Nominal by Nominal	Phi	,505	,226	,209
	Cramer's V	,357	,226	,209
N of Valid Cases		32		

Conclusion: It is not possible to reject null hypothesis on the significance level 0.05. According to Fisher´s Exact test both variables are independend.

4.6.7 Size of a team * Scope

H_0: Size of a team that was used in a project is independent on the success perception of the scope of the project.

H_A: Meeting proper scope of the project depends on the size of the team working on the project.

The estimation is that size of a team that is chosen will have an influence on delivering project meeting all requirements and objectives.

Crosstab

Count

		Scop			
		2	3	4	Total
Size of a team	<3	3	7	2	12

	3<6	1	11	6	18
	12<50	0	1	1	2
Total		4	19	9	32

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)
Pearson Chi-Square	3,559[a]	4	,469	,514
Likelihood Ratio	3,676	4	,452	,554
Fisher's Exact Test	3,690			,498
N of Valid Cases	32			

a. 6 cells (66,7%) have expected count less than 5. The minimum expected count is ,25.

Symmetric Measures

		Value	Approx. Sig.	Exact Sig.
Nominal by Nominal	Phi	,334	,469	,514
	Cramer's V	,236	,469	,514
N of Valid Cases		32		

Conclusion: It is not possible to reject null hypothesis on the significance level 0.05. According to Fisher´s Exact test both variables are independend.

4.6.8 Size of a team * Cost

H_0: Size of a team that was used in a project is independent and has nothing to do with the success perception of delivering project within estimated cost.
H_A: Meeting estimated cost of the project depends on the size of the team working on the project.
The estimation is that size of a team that is chosen will have an influence on delivering project meeting estimated cost.

Crosstab

Count

		Cost			
		3	4	5	Total
Size of a team	<3	2	4	6	12
	3<6	3	8	7	18
	12<50	0	1	1	2
Total		5	13	14	32

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)
Pearson Chi-Square	,821[a]	4	,936	,957
Likelihood Ratio	1,136	4	,888	,957
Fisher's Exact Test	1,257			,957
N of Valid Cases	32			

a. 6 cells (66,7%) have expected count less than 5. The minimum expected count is ,31.

Symmetric Measures

		Value	Approx. Sig.	Exact Sig.
Nominal by Nominal	Phi	,160	,936	,957
	Cramer's V	,113	,936	,957
N of Valid Cases		32		

Conclusion: It is not possible to reject null hypothesis on the significance level 0.05. According to Fisher´s Exact test both variables are independend. There is no connection within these two variables.

4.7 Success Factors Statistics

There was 20 questions in the survey to determine success factors of agile project management. These questions were divided into 4 groups. The goal was to determine, whether there is a difference between groups in success perception of the agile project. 5-point Likert scale was used in a survey.

List of questions:

Perception	(scale 1-5)
Category	Question
Organization	1. Strong executive support
	2. Committed sponsor or manager
	3. Cooperative organizational culture instead of hierarchal
	4. Oral culture placing high value on face-to-face communication
	5. Organizations where agile methodology is universally accepted
People	6. Team members with high competence and expertise
	7. Team members with great motivation
	8. Managers knowledgeable in agile process
	9. Good customer relationship
	10. Reward system appropriate for agile
Process	11. Following agile-oriented project management process
	12. Strong communication focus with daily face-to-face meetings
	13. Honoring regular working schedule – no overtime
	14. Strong customer commitment and presence
	15. Customer having full authority

Technical	16. Well-defined coding standards up front
	17. Pursuing simple design
	18. Right amount of documentation
	19. Regular delivery of software
	20. Delivering most important features first

Descriptive Statistics

	N	Mean	Std. Deviation	Minimum	Maximum
Perception	20	3,1690	,91056	1,47	4,41

Kruskal-Wallis Test

The Kruskal-Wallis Test was used to determine if there is a statistical difference in median of these groups. The simple average was used to compute average value in each question.

Ranks

	Group	N	Mean Rank
Perception	1	5	5,40
	2	5	18,00
	3	5	10,60
	4	5	8,00
	Total	20	

Test Statistics[a,b]

	Perception
Chi-Square	12,684
Df	3

Asymp. Sig.	,005

a. Kruskal Wallis Test

b. Grouping Variable: Group

Ho: There is no statisticly significant difference in median between groups.

Ha: There are at least two groups that are statisticly different in media.

Result: According to Kruscal-Wallis test the null hypothesis is rejected on the significance level 0,05, there is difference in means between at least two groups of questions.

Neményi method was used to distinguish between which groups is the difference.

Neményi Critical Value = 93,4			
	People	Process	Technical
Organization	**181,5**	87,6	54,3
People		76,4	**130**
Process			56,3

As we can see there is statisticly significant difference mostly between people.

To conclude, we can summarize that respondes state that the people are critical factor of success perception of the agile project management. Generally, the result was expected, people are the most difficult to manage and have a lot of uncertainty.

5 Case Study

This case study is conducted to study a software development project inside a company. First, the organization will be described, second the product and project management. The analysis based on agile framework will be conducted.

5.1 Organization

The case study is about a venture of an organization. The goal of the venture was not just to deliver a software but also a marketing strategy or to take care about legal issues. This was viewed by the managers of the parent company is rather special and very uncommon for their style of managing daily issues. Parent company provided help with some resources. Like facilities, hardware, software, marketing material and mainly financial resources.

5.2 Team

The team consisted of 5 people. One person was a Project Leader, second managed product lunch activites. There were also 3 software developers.

5.3 Product

The final product was a Microsoft .NET based connector to proprietary Czech solution of delivering letters electronically – “Datové schránky”. There were two main possible distributing channels. First, you could download the connector “as is” and implement it by your own IT department. Second, there was a possibility to order the complete solution with implementation services delivered by parent company.

5.4 Communication

All of the team members worked with each other before. The team was working in an open workspace, so communication was very easy to start. Weekly meetings were held. Instead of writing documents and emails, the team members started to implement the discussed issues right away.

Besides face-to-face communication mobile phones and Skype was often used when working homeoffice.

5.5 Product Development

The team started the product development from scratch, and they did not have any formal, documented process to follow. A project plan was set according to the external market requirements. From 1.11.2009 there should be a start of the "Datové schránky" so they must to fulfill the date of delivery the Connector. The working methods of team members arose from the previous projects they had before, thus they had an unwritten, implicit process model based on their tacit knowledge to follow instead of a formal process.

5.6 Project Management

The business decision of develepoment of the project was done. Rough estimate about the budget and the schedule for the project was stated. The main goal was to fulfill the date of delivery at the beginning of the November 2009.

5.7 Change Management

No special requirements management software system was used. Intead, excel-sheets fro capturing and communication requirements throught the project was used. These excel-sheets were cisculated among the involved stakeholdes one at

a time, and they added new requirements and feature proposals to the file. Then it was sent to the next stakeholder and finally back to the venture team.

5.8 Implementation

Firstly, the prototype wan developed. Functionality came along as the development work went on. A new build of the product was finished in the end of eash iteration round.

5.9 Testing

Usability checks and systems test were made for the procuct in different phases of product development. Usability checking did another deparment of the parent corporation.

5.10 Summary

As the case analysis reveals, the values and principles of Agile methods were imperceptible. This confirms that the development of the project is very much based on practices and ways or methods of working that experienced people have from the past. Those practices arise from tacic knowledge and intuition if the circumstances are favourable.

6 Recommendations

In spite of the fact that the agile approach was required from the management of the enterprise the agile approach was not delivered. SCRUM method could be used effectively. The requirements from the external environment, mainly from the Česká pošta, s.p. (Czech Post) were changing vastly. Furthermore, the parent company changed few time the specification of their systems where the Connector shoud be used.

Traditional and intuitive approcha should have come to behind. It could be better to use for such a project on of the Agile Methods, SCRUM should have been considered. The iteration and incremental framework should be used rather that intuitive principles.

Finally, there was no system or clear defined process of change management. On the market there is a lot of software providing such functionality. Some of them are even uner Open Source Licence.

7 Conclusions

The first goal of this thesis was to explain two mainly used method of Agile Project Management. In the beginning the complexity issues was discussed. Dimensions of complexity were defined. Because of the complexity the decentralization of control and commitment is required these days. The traditional methods of project management do not satisty the fast changing environment. As a response the Agile methods arrised after the year 2000.

The SCRUM method was described first. The process consisting of several parts were described. The SCRUM is nothing more that emphasis on communication and usage fo common sense.

Second, the Extreme Programming method was described. This method is mainly used for software development.

The survey among professionals with experience with Agile project management was conducted with emphasis on the perception of the success of their projects. Success attributes were defined as time, scope, cost and quality. The 5 point Likert scale was used to measure. As the research shown the project had the level of Quality seen from the responders on the average and below the average. The attribute Time as considered below average. Interestingly almost all the projects examined met Cost criterium. The scope of the project was reached on average and above the average.

Furthermore, the crosstab analysis was used to be able to say whether there is a statistical significant difference among methods that was used in a projects and success factors (time, scope, quality, cost) as well as the size of the project team and success factors. The method comparing to time and method comparing to cost shows statisticly significant difference. We can sum up that method used in the project has a significant impact on the success of the project.

Next, the success factors analysis was held. Using statistics there was proven the estimation that the people have significant impact on the success of the project.

The case study was conducted and several recommendations were made.

As a proposition for extend of the paper all possible methods not just SCRUM and XP should be described. Similary all these methods could be used it the analogous survey. Comparing these methods with regard to the success of projects could be a vast simplification for every project manager to decide which method he or she should use.

8 Bibliography

1. **Merriam-Webster.** WWWebster Dictionary and Thesaurus . [Online] 2002. [Citace: 05. 01 2010.] http://www.m-w.com/home.htm.

2. **Highsmith, Jim.** *What is Agile Software Development.* [PDF] Flagstaff : Cutter Consorcium, 2003.

3. **Schwaber, Ken.** *Agile Project Management with Scrum.* Washington : Microsoft Press, 2003. 0-7356-1993-X.

4. **Cohn, Mike.** *Introducing an Agile Process to an Organization.* [PDF] s.l. : IEEE, 2003.

5. **Dyba, Tore a Dingsøyr, Torgeir.** *Empirical studies of agile software development: A systematic review.* [PDF] Trondheim, Norway : SINTEF ICT, 2008.

6. **Škoda, Ondřej.** *Agilní metodiky vývoje SW - Diplomová práce.* Brno : Masarykova Univerzita, Fakulta Informatiky, 2009.

7. **Nerur, Shridhar a VenuGopal, Balijepally.** *Theoretical Reflections on AGILE DEVELOPMENT METHODOLOGIES.* [PDF] místo neznámé : Communications of the ACM, 2007.

8. **Lindstrom, Lowell and Jeffries, Ron.** *Extreme Programming and Agile Software Development Methodologies.* [PDF] s.l. : Information Systems Management Summer, 2004.

9. **Abrahamssona, Pekka, a další.** *New Directions on Agile Methods: A Comparative Analysis.* [PDF] Oulu : Technical Research Center of Finland, VTT Electronics, 2003.

10. **Krasteva, Iva.** *Adopting an Agile Methodology — Why It Did Not Work.* [PDF] Sofia : Sofia University, 2004.

11. **Cohen, David, Lindvall, Mikael a Costa, Patricia.** *Agile Software Development.* [PDF] Maryland : The University of Maryland, 2003.

12. **Tsun Chow, Dac-Buu Cao.** *A survey study of critical success factors in agile software projects.* [PDF] Minneapolis : The Journal of Systems and Software, 2008.

13. **Weisstein, Eric.** Fisher's Exact Test. *MathWorld.* [Online] A Wolfram Web Resource., 2008. [Citace: 12. 01 2010.] http://mathworld.wolfram.com/FishersExactTest.html.

14. Cramer´s V. *Cramer´s V.* [Online] 2008. [Citace: 1. 02 2010.] http://myyn.org/m/article/cramers-v/.

15. **Helena Holmström, Brian Fitzgerald, Pär J. Ågerfalk, and Eoin Ó. Conchúir.** *Agile Practices Reduce Distance in Global Software Development.* [PDF] místo neznámé : Information Systems Management, 2006.

16. **Cohn, Michael.** *Agile Estimating an Planning.* [PDF] s.l. : Prentice Hall, 2006.

17. **Tekinerdoğan, Bedir, a další.** *Aspect-Oriented Requirements Engineering and Architecture Design.* [PDF] Lancaster : Early Aspects, 2004.

18. **Kalermo, Jonna and Rissanen, Jenni.** *Agile Software Development.* [PDF] Jyvaskyla : University of Jyvaskyla, 2002.

19. **Paasivaara, Maria, Durasiewicz, Sandra a Lassenius, Casper.** *Using Scrum in a Globally Distributed Project: A Case Study.* [PDF] Helsinky : Software Business and Engineering Institute, Helsinki University of Technology, 2008.

20. **Vymětal, David.** *Information systems projects in companies and their implementation.* [PDF] Karviná : Silezian Univerzity in Opava, School of Business, 2008.

9 Supplements

9.1 Success Factors Statistics

Statistics

		1. Strong executive support	2. Committed sponsor or manager	3. Cooperative organizational culture instead of hierarchal	4. Oral culture placing high value on face-to-face communication
N	Valid	32	32	32	32
	Missing	0	0	0	0
	Mean	1,91	1,47	2,94	3,53
	Std. Error of Mean	,164	,090	,179	,090
	Median	2,00	1,00	2,00	4,00
	Mode	2	1	2	4
	Std. Deviation	,928	,507	1,014	,507
	Variance	,862	,257	1,028	,257
	Skewness	,969	,131	,131	-,131
	Std. Error of Skewness	,414	,414	,414	,414
	Kurtosis	,378	-2,119	-2,119	-2,119
	Std. Error of Kurtosis	,809	,809	,809	,809
	Minimum	1	1	2	3
	Maximum	4	2	4	4
	Sum	61	47	94	113

Percentiles	25	1,00	1,00	2,00	3,00
	50	2,00	1,00	2,00	4,00
	75	2,00	2,00	4,00	4,00

Statistics

		5. Organizations where agile methodology is universally accepted	7. Facility with proper agile-style work environment	8. Reward system appropriate for agile	9. Team members with high competence and expertise
N	Valid	32	32	32	32
	Missing	0	0	0	0
	Mean	1,63	3,00	2,00	3,41
	Std. Error of Mean	,117	,000	,000	,167
	Median	2,00	3,00	2,00	3,00
	Mode	1	3	2	3
	Std. Deviation	,660	,000	,000	,946
	Variance	,435	,000	,000	,894
	Skewness	,584			,288
	Std. Error of Skewness	,414	,414	,414	,414
	Kurtosis	-,570			-,709
	Std. Error of Kurtosis	,809	,809	,809	,809
	Minimum	1	3	2	2
	Maximum	3	3	2	5
	Sum	52	96	64	109

Percentiles	25	1,00	3,00	2,00	3,00
	50	2,00	3,00	2,00	3,00
	75	2,00	3,00	2,00	4,00

Statistics

		10. Team members with great motivation	11. Managers knowledgeable in agile process	12. Good customer relationship	13. Following agile-oriented project management process
N	Valid	32	32	32	32
	Missing	0	0	0	0
	Mean	4,28	4,41	3,47	3,75
	Std. Error of Mean	,197	,190	,180	,119
	Median	5,00	5,00	4,00	4,00
	Mode	5	5	4	4
	Std. Deviation	1,114	1,073	1,016	,672
	Variance	1,241	1,152	1,031	,452
	Skewness	-1,050	-1,410	-1,093	-2,381
	Std. Error of Skewness	,414	,414	,414	,414
	Kurtosis	-,619	,317	,576	3,909
	Std. Error of Kurtosis	,809	,809	,809	,809

	Minimum	2	2	1	2
	Maximum	5	5	5	4
	Sum	137	141	111	120
Percentiles	25	3,00	3,50	3,00	4,00
	50	5,00	5,00	4,00	4,00
	75	5,00	5,00	4,00	4,00

Statistics

		14. Strong communication focus with daily face-to-face meetings	15. Honoring regular working schedule – no overtime	16. Strong customer commitment and presence	17. Customer having full authority
N	Valid	32	32	32	32
	Missing	0	0	0	0
	Mean	3,78	2,94	2,72	3,06
	Std. Error of Mean	,184	,179	,136	,179
	Median	4,00	2,00	3,00	4,00
	Mode	4	2	2	4
	Std. Deviation	1,039	1,014	,772	1,014
	Variance	1,080	1,028	,596	1,028
	Skewness	-,635	,131	,546	-,131
	Std. Error of Skewness	,414	,414	,414	,414

	Kurtosis	-,653	-2,119	-1,081	-2,119
	Std. Error of Kurtosis	,809	,809	,809	,809
	Minimum	2	2	2	2
	Maximum	5	4	4	4
	Sum	121	94	87	98
Percentiles	25	3,00	2,00	2,00	2,00
	50	4,00	2,00	3,00	4,00
	75	4,75	4,00	3,00	4,00

Statistics

		18. Well-defined coding standards up front	19. Pursuing simple design	20. Right amount of documentation	
N	Valid	32	32	32	
	Missing	0	0	0	
	Mean	3,16	2,94	1,72	
	Std. Error of Mean	,128	,179	,112	
	Median	3,00	2,00	2,00	
	Mode	3	2	2	
	Std. Deviation	,723	1,014	,634	
	Variance	,523	1,028	,402	
	Skewness	-,248	,131	,301	

	Std. Error of Skewness	,414	,414	,414	
	Kurtosis	-,981	-2,119	-,556	
	Std. Error of Kurtosis	,809	,809	,809	
	Minimum	2	2	1	
	Maximum	4	4	3	
	Sum	101	94	55	
Percentiles	25	3,00	2,00	1,00	
	50	3,00	2,00	2,00	
	75	4,00	4,00	2,00	

Frequency Table

1. Strong executive support

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	12	37,5	37,5	37,5
	2	14	43,8	43,8	81,3
	3	3	9,4	9,4	90,6
	4	3	9,4	9,4	100,0
	Total	32	100,0	100,0	

2. Committed sponsor or manager

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	17	53,1	53,1	53,1
	2	15	46,9	46,9	100,0
	Total	32	100,0	100,0	

3. Cooperative organizational culture instead of hierarchal

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	17	53,1	53,1	53,1
	4	15	46,9	46,9	100,0
	Total	32	100,0	100,0	

4. Oral culture placing high value on face-to-face communication

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	3	15	46,9	46,9	46,9
	4	17	53,1	53,1	100,0
	Total	32	100,0	100,0	

5. Organizations where agile methodology is universally accepted

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	15	46,9	46,9	46,9
	2	14	43,8	43,8	90,6
	3	3	9,4	9,4	100,0
	Total	32	100,0	100,0	

7. Facility with proper agile-style work environment

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	3	32	100,0	100,0	100,0

8. Reward system appropriate for agile

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	32	100,0	100,0	100,0

9. Team members with high competence and expertise

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	5	15,6	15,6	15,6
	3	14	43,8	43,8	59,4
	4	8	25,0	25,0	84,4
	5	5	15,6	15,6	100,0
	Total	32	100,0	100,0	

10. Team members with great motivation

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	3	9,4	9,4	9,4
	3	7	21,9	21,9	31,3
	5	22	68,8	68,8	100,0
	Total	32	100,0	100,0	

11. Managers knowledgeable in agile process

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	3	9,4	9,4	9,4
	3	5	15,6	15,6	25,0

	5	24	75,0	75,0	100,0
	Total	32	100,0	100,0	

12. Good customer relationship

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	2	6,3	6,3	6,3
	2	4	12,5	12,5	18,8
	3	5	15,6	15,6	34,4
	4	19	59,4	59,4	93,8
	5	2	6,3	6,3	100,0
	Total	32	100,0	100,0	

13. Following agile-oriented project management process

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	4	12,5	12,5	12,5
	4	28	87,5	87,5	100,0
	Total	32	100,0	100,0	

14. Strong communication focus with daily face-to-face meetings

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	6	18,8	18,8	18,8
	3	3	9,4	9,4	28,1
	4	15	46,9	46,9	75,0
	5	8	25,0	25,0	100,0
	Total	32	100,0	100,0	

15. Honoring regular working schedule – no overtime

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	17	53,1	53,1	53,1
	4	15	46,9	46,9	100,0
	Total	32	100,0	100,0	

16. Strong customer commitment and presence

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	15	46,9	46,9	46,9
	3	11	34,4	34,4	81,3

	4	6	18,8	18,8	100,0
	Total	32	100,0	100,0	

17. Customer having full authority

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	15	46,9	46,9	46,9
	4	17	53,1	53,1	100,0
	Total	32	100,0	100,0	

18. Well-defined coding standards up front

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	6	18,8	18,8	18,8
	3	15	46,9	46,9	65,6
	4	11	34,4	34,4	100,0
	Total	32	100,0	100,0	

19. Pursuing simple design

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	2	17	53,1	53,1	53,1
	4	15	46,9	46,9	100,0
	Total	32	100,0	100,0	

20. Right amount of documentation

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	12	37,5	37,5	37,5
	2	17	53,1	53,1	90,6
	3	3	9,4	9,4	100,0
	Total	32	100,0	100,0	