# Agile Project Estimation

Audrius Grinys

Faculty of Industrial Engineering,
Mechanical Engineering and
Computer Science
University of Iceland
2012

# Agile Project Estimation

Audrius Grinys

30 ECTS thesis submitted in partial fulfillment of a
*Magister Scientiarum* degree in Computer Science

Advisor(s)
Helgi Þorbergsson
Hjálmtýr Hafsteinsson

Faculty Representative
Páll Melsted

Faculty of Industrial Engineering, Mechanical Engineering and Computer Science
School of Engineering and Natural Sciences
University of Iceland
Reykjavík, May 2012

Agile Project Estimation
Goal of the thesis is to predict project estimation based on a given estimation from a developer by user stories and to make the estimate closer to actual time as possible.
30 ECTS thesis submitted in partial fulfillment of a *Magister Scientiarum* degree in Computer Science

Faculty of Industrial Engineering, Mechanical Engineering and Computer Science
School of Engineering and Natural Sciences
University of Iceland
Hjarðarhagi 2 – 6
107, Reykjavík
Iceland

Telephone: 525 4000

Bibliographic information:
Audrius Grinys, 2012, *Agile Project Estimation*, Master’s thesis, Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland, pp. 79.

Printing: Háskólaprent, Fálkagata 2, 107 Reykjavík

Reykjavík, Iceland, May 2012

# Abstract

Agile software development is an iterative development method. Its basic concept is people – centered and as such, it acknowledges that requirements can change. Estimation and structure of agile methodologies are very different from those in traditional ones. However, research involving estimation in agile methodologies is considerably less advanced. The iterative based software development methods in agile methodologies are powerful ways to deliver high quality software on time. To ensure the quality, the impact of factors affecting the development cycle should be evaluated constantly.

This thesis introduces a detailed view of user stories and how user requirements are transformed into the right format as user stories. The risk and value of given requirements is discussed and how changeability of a user story is defined and applied to the estimation process.

Changeability rate of the user story helps to define estimation more accurately using probability of the possible change of the user story. Main indicators of the changeability rate are the risk and value of the user story and evaluation of the INVEST model of the user story.

**Keywords:** Agile methodologies, Scrum development methodology, agile estimation, user story estimation, planning poker, changeability rate.

# Útdráttur

Kvik hugbúnaðargerð er ítrunarferli sem snýst um fólk og síbreytilegt umhverfi og tekur tillit til þeirrar staðreyndar að þarfir breytast. Uppbygging og umfangsmat kvikrar hugbúnaðargerðar er talvert ólík þeirri sem tíðkast í hefðbundnari aðferðum. Hins vegar hafa mun færri rannsóknir verið gerðar á umfangsmati í kvikri hugbúnaðargerð. Ítrunarferlið í kvikri hugbúnaðargerð er öflug aðferð til að þróa hágæða hugbúnað sem afhentur er á réttum tíma. Þeir þættir sem hafa áhrif á hugúnaðargerðina þurfa að vera endurmetnir reglulega til að tryggja gæði hugbúnaðarins.

Þessi ritgerð fjallar ítarlega um notendasögur og hvernig þörfum notenda er lýst sem sögum á réttu formi. Áhætta og ávinningur þarfa er kynntur og breytanleiki notendasaga er skilgreindur og notaður til umfangsmats.

Breytanleiki notendasagna auðveldar að gera nákvæmara umfangsmat sem byggir á líkum þess að notendasaga taki breytingum.  Helstu þæættir breytanleika eru áhætta og ávinnungur notendasögu og beiting INVEST líkansins á notendasögu.

**Lykilorð:** Kvik hugbúnaðargerð, Scrum – aðferðafræði, kvikt umfangsmat, umfangsmat notendasagna, áætlunarpóker, breytanleiki.

*Dedication*

*To my wonderful daughter Gustė*

# Table of Contents

**List of Figures .......... x**

**List of Tables .......... xi**

**Abbreviations .......... xii**

**Acknowledgements .......... xiii**

**1 Introduction .......... 1**
1.1 Area of Research .......... 1
1.2 Research Problem .......... 2
1.3 Research Goals .......... 2
1.4 Structure of Document .......... 2

**2 Agile Development .......... 4**
2.1 Agile Manifesto .......... 6
2.2 Agile Projects .......... 6
2.2.1 Scrum projects .......... 7
2.2.2 Why Scrum .......... 8
2.3 Summary .......... 8

**3 User Story .......... 10**
3.1 User Stories, Epics and Themes .......... 10
3.1.1 Agile Themes .......... 10
3.1.2 Agile Epics .......... 10
3.1.3 Agile User Stories .......... 11
3.2 Format of User Story .......... 11
3.2.1 Card .......... 11
3.2.2 Conversation .......... 11
3.2.3 Confirmation .......... 12
3.3 Requirements for the User story .......... 12
3.3.1 Independent .......... 12
3.3.2 Negotiable .......... 13
3.3.3 Valuable .......... 13
3.3.4 Estimable .......... 13
3.3.5 Small .......... 14
3.3.6 Testable .......... 14
3.3.7 Evaluation INVEST model .......... 14
3.4 Prioritizing user stories .......... 16
3.5 Changeability of the User Story .......... 18
3.6 Procedures and Work Instructions .......... 22

3.6.1 Procedures ........ 22
3.6.2 The Five W's ........ 24
3.6.3 Meeting Work Instructions ........ 25
3.7 Recognize Bad user story ........ 27
3.8 Tools ........ 28
3.8.1 JIRA (GreenHopper plugin) ........ 29
3.8.2 Mingle ........ 29
3.8.3 Scrumy ........ 29
3.8.4 ProjectCards ........ 29
3.8.5 Trello ........ 30
3.9 Summary ........ 30

**4 Velocity ........ 32**

**5 Estimating User Story ........ 36**
5.1 Prepare for the Estimation ........ 36
5.1.1 Powers of two ........ 36
5.1.2 No averages or numbers not on the scale ........ 37
5.1.3 Vote independently ........ 37
5.1.4 Take the largest estimate ........ 37
5.1.5 Large estimate gaps ........ 37
5.1.6 Insufficient information ........ 38
5.1.7 Estimation group size ........ 38
5.1.8 No laptops ........ 38
5.1.9 Required participation ........ 38
5.1.10 Stale estimations ........ 38
5.1.11 Bribes ........ 39
5.2 Planning poker ........ 39
5.2.1 The Moderator ........ 40
5.2.2 Participants ........ 41
5.2.3 The Five Steps ........ 41
5.2.4 When to Play Planning Poker ........ 43
5.2.5 Why Planning Poker Works ........ 43
5.3 Summary ........ 44

**6 Agile Estimation ........ 45**
6.1 Agile estimation workflow ........ 47
6.2 Successful estimation ........ 48
6.3 Expert Opinion ........ 50
6.4 Analogy ........ 50
6.5 Disaggregation ........ 50
6.6 Summary ........ 52

**7 Conclusions ........ 53**
7.1 Summarizing ........ 53
7.2 Future work ........ 54

**References** ........................................................................ **55**

**8 Appendix – Case Study** ........................................................ **59**

8.1 User stories ........................................................................ 59
8.2 INVEST model evaluation ..................................................... 61
8.3 Risk and value evaluation ..................................................... 62
8.4 Changeability rate ................................................................ 63

# List of Figures

Figure 1. Scrum workflow ........ 5

Figure 2. Kanban workflow ........ 5

Figure 3. User story workflow ........ 11

Figure 4. User stories' distribution by risk and value ........ 17

Figure 5. Quadrants for prioritizing user stories ........ 18

Figure 6. Changeability quadrants by risk and value ........ 19

Figure 7. User story’s definition and estimation workflow ........ 23

Figure 8. Team's velocity ........ 33

Figure 9. Planning Poker overview ........ 39

Figure 10. Agile estimation overview ........ 45

Figure 11. Agile estimating workflow ........ 47

Figure 12. User stories' distribution by RV [case study] ........ 62

# List of Tables

Table 1. Scrum basics ........ 7

Table 2. INVEST model evaluation and metrics ........ 14

Table 3. Evaluation example for INVEST model ........ 15

Table 4. Metrics for the prioritizing user stories ........ 16

Table 5. Prioritizing user stories by risk and value ........ 17

Table 6. Example values for the changeability rate ........ 20

Table 7. Changeability rate ........ 21

Table 8. Meeting work instructions ........ 25

Table 9. Agile project management tools ........ 29

Table 10. Velocity by iteration ........ 32

Table 11. Product backlog ........ 48

Table 12. User stories [case study] ........ 59

Table 13. Evaluation of INVEST model [case study] ........ 61

Table 14. Risk and value evaluation [case study] ........ 62

Table 15. Changeability rate [case study] ........ 63

# Abbreviations

ASD – Agile Software Development

ChR – Changeability Rate

CSP – Completed Story Points

DSDM – Dynamic System Development Model

$I_{length}$ – Iteration length in days that took the team to finish this work

INVEST – Independent, Negotiable, Valuable, Estimable, Small, Testable

QA – Quality Assurance

R – Risk defined of user story

RV – Risk and Value

SP – Story Points

TDD – Test Driven Development

US – User Story

V – Value defined of user story

$V_{team}$ – Velocity of the team

XP – eXtreme Programing

# Acknowledgements

This M.Sc. project was carried out at the Faculty of Industrial Engineering, Mechanical Engineering and Computer Science at the University of Iceland.

I would like to express my gratitude to all the individuals and organizations involved that enabled and facilitated this master thesis.

Special thanks goes to my supervisor, Associate Professor Dr. Helgi Þorbergsson, for his tireless support through scope definition, experience sharing and continuous motivation.

To my co – supervisor, Associate Professor Hjálmtýr Hafsteinsson, for important guidance and motivation.

To Hafþór Pálsson, for his detailed review of the thesis and valuable feedback.

To all my colleagues who accompanied me throughout the time with constant support and guidance.

To my family, for their understanding, care, and for being the major reason behind my focus on academic investigation.

To my friends, for their constant help and support.

Reykjavík, 16. May 2012

Audrius Grinys

# 1 Introduction

The planning of traditional software development methods is conventional; it usually first defines a specific project scope, and then does the top – down function decomposition based on the scope, and finally estimates development time and development costs. Traditional planning methods often cannot balance scope, duration, cost and quality, and problems always arise. Detailed investigations are difficult to undertake and determine all the users' needs only once, integration and testing are always left at last, these conditions usually cause budget overrun, eventually leading to the project's failure (Charette, 2005). In the Agile development, one big software project is decomposed into multiple executable sub – projects. Start by finishing the most important functions that are selected by users. They use iterative incremental development methods, thus each iteration results in a running system. Development team focuses on quickly dealing with the changing needs.

## 1.1 Area of Research

Enterprises usually achieve innovation through creative software development, despite the associated risk within the very abstract process of software development (Kenneth, 2009).

Estimating the resources of a software development project is one of the most crucial tasks for project managers but despite this it continues to be a weak link in the software development field (Kenneth, 2009).

Why do we estimate?

- To Plan —When will something be done?
- To Schedule —What order should we do things in?
- To Hire —Do we need more people to do the work?
- To Price —How much will it cost?
- To Guide Investment —Is doing something worth it?

According to a CHAOS Report (Standish Group, 2009), about 70% of IT development projects fail to deliver functional software, mostly due to poor communication between stakeholders who play key roles in the development process. The IT software industry has been in crisis as time and cost overruns are mostly caused by human factors in software development collaboration.

The software development process, regardless of the methodology adopted, requires effective management and planning. A large part of this planning is the creation of estimates at the beginning of a project so that resources can be appropriately allocated. Estimation techniques and models are available to simplify this activity but the prevalence of cost and schedule overruns on software development projects indicates that accurate estimation remains somewhat elusive.

## 1.2 Research Problem

Agile projects use an iterative process to realize incremental delivery. One of the main principles of agile methods is to “welcome changing requirements”, even though changing requirements are a major cause of software cost estimating problems (Beck et al., 2001).

The main factors that are typically estimated at the beginning of a software development project are: cost, size, schedule, people resources, quality, effort, resources, maintenance costs, and complexity.

The main concerns that induced this research and identified problems are:

- Prepare correct user stories using INVEST model;
- Prioritize user stories;
- Evaluate risk and value, be ready to accept changing requirements in any phase of an iteration;
- Make an estimation based on these dynamic circumstances;
- What estimation techniques to use for the estimating user stories?
- How estimation fits into iteration plan?

## 1.3 Research Goals

In order to answer the problems identified in this research, this thesis aims to review the related work and present a proposal for the identified problem as well as to evaluate it.

Scrum, which is a popular and widely adopted agile methodology, was chosen as the target of the proposed estimation model.

Thus, the goal of the thesis is to predict project estimation based on given estimation from developer by user stories and to make it closer to actual time as possible.

## 1.4 Structure of Document

Thesis consists of 5 major parts. Chapter 2 describes an overview of agile development. This section provides an overview of agile development methods such as Scrum and its main processes. Chapter 3 describes user requirements as User Stories. This part is one of the most important one. It looks at collecting user requirements into user stories, evaluating them from risk and value perspectives, prioritizing them into iterations. Chapter 4 considers velocity of the team. What is velocity in agile teams? How velocity is calculated? Influences of the velocity in estimation. Chapter 5 describes estimating user stories. This part considers estimating a user story using the “Planning Poker” approach. How to use this approach to estimate? Why does it working? When to apply planning poker in agile development? Chapter 6 gives an overview of the whole agile estimation.

What approaches are most popular, what are their differences and usage? To conclude Chapter 7 will present some final words and considers possible future work.

# 2 Agile Development

*"These days we do not program software module-by-module, we program software feature-by-feature."*–Mary Poppendieck

Agile software development is a group of software development methodologies based on iterative and incremental development models, where solutions and requirements evolves through collaboration between teams. It relates with adaptive planning, a time – box iterative approach, evolutionary development and delivery, accessible rapid and flexible response to change. Agile development methodology attempts to provide many opportunities to assess the direction of a project throughout the development lifecycle. This is achieved through sprints or iterations, at the end of which teams must present a shippable increment of work (Larman, 2003).

Some methods of agile development:

- Scrum
- XP (eXtreme Programing)
- Kanban
- DSDM (Dynamic System Development Model)
- Crystal Clear

The main differences between these methodologies are the rules and workflow. Rules provides on how the work should be performed. For example in Scrum we could have a lot of rules, some of them could be (Coutinho, 2010):

1. The team's work could not be interrupted during the sprint;
2. The team work is time – boxed;
3. Teams do a demo to stakeholders at the end of each sprint;

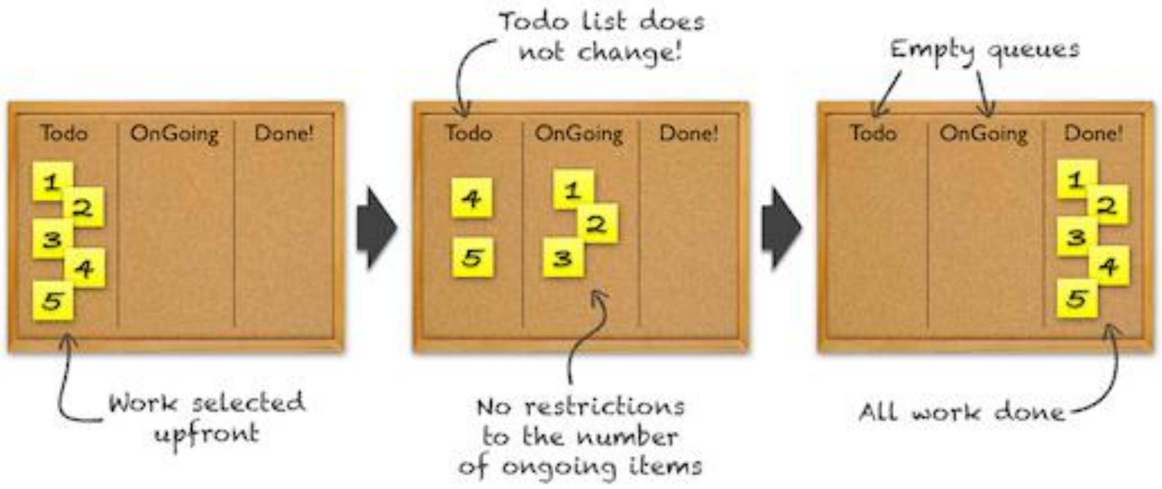
   And so on…

Compared with Kanban development method which is more open than Scrum, and it has only couple of rules (Coutinho, 2010):

1. Visualize your workflow;
2. Limit your Work In Progress.

Comparing these two methodologies from workflow point, the main difference is that in Scrum sprint has an "end" but Kanban is "endless".

In Scrum, you select the work you will be doing for the next sprint beforehand. You then lock the sprint, do all the work, and after a couple of weeks – the usual sprint duration – your queue is empty (Schwaber, 2010).



Todo list does not change!
Empty queues
Todo
OnGoing
Done!
1
2
3
4
5
Work selected upfront
No restrictions to the number of ongoing items
All work done

**Figure 1. Scrum workflow**

In Kanban, all that's limited is the size of the queues, called the Work In Progress limit. This means that you can change the items in the queues at any time, and that there's no "sprint end". The work just keeps flowing (Coutinho, 2010).

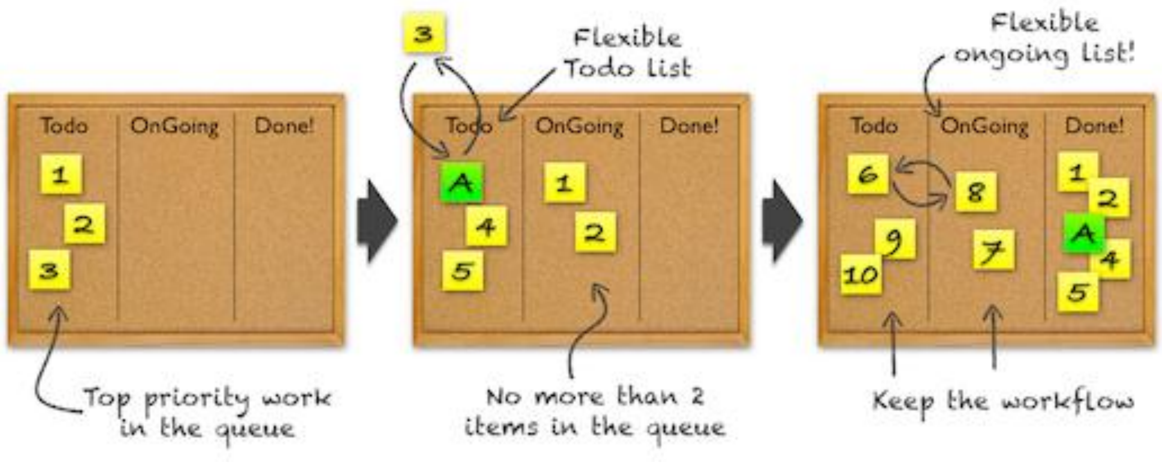


**Figure 2. Kanban workflow**

DSDM is probably the original and the most complete agile development method. DSDM was around before the term ‘agile’ was even invented, but is absolutely based on all the principles we’ve come to know as agile.

XP is a more radical agile methodology, focusing more on the software engineering process and addressing the analysis, development and test phases with novel approaches that make a substantial difference to the quality of the end product.

## 2.1 Agile Manifesto

Twelve principles underlie the Agile Manifesto, including (Beck et al., 2001):

- Customer satisfaction by rapid delivery of useful software;
- Welcome changing requirements, even late in development;
- Working software is delivered frequently (weeks rather than months);
- Working software is the principal measure of progress;
- Sustainable development, able to maintain a constant pace;
- Close, daily co – operation between business people and developers;
- Face-to-face conversation is the best form of communication (co – location);
- Projects are built around motivated individuals, who should be trusted;
- Continuous attention to technical excellence and good design;
- Simplicity;
- Self-organizing teams;
- Regular adaptation to changing circumstances.

## 2.2 Agile Projects

Agile project planning basically is feature based then traditional project is task based. Project planning usually happens something like this:

- Gather all of the requirements.
- Perform analysis work on all of the required functionality
- Perform design work on all of the required functionality
- Develop code for all of the required functionality
- Test all of the required functionality
- Deploy all of the required functionality.

Agile project plans are organized into time bound iterations, usually anywhere from 2 – 4 weeks in length. During this iteration all necessary work to take features from an idea to working product is completed. The end result is that portions of the product are delivered on a regular, frequent basis (after each iteration). This gives stakeholders a much better idea of the state of the project, because they can see and use the end result as it becomes available (Misra and Kumar, 2009).

### 2.2.1 Scrum projects

In this thesis Scrum development processes will be used and will look from this perspective to the project planning. Scrum is a lightweight process that can manage and control software and product development: it is a project management process. Scrum embraces iterative and incremental practices. There are three key roles in a Scrum (Schwaber, 2001):

1. Scrum Master;
2. Product Owner;
3. Development Team / Project Team.

The Scrum Master is key, he or she "represents management to the project". Such a role is usually filled by a Project Manager or Team Leader. They are responsible for enacting Scrum values and practices. Their main job is to remove impediments, i.e. project issues that might slow down or stop activity that moves the project forward.

The Product Owner is possibly a Product Manager or Project Sponsor, a member of Marketing or an Internal Customer.

The Project Team should consist of 5-10 members. The team itself should be cross – functional, involving individuals from QA, Programmers, UI Designers, etc.

From the table below we can see how Scrum ensures that the project remains active and continually moves forward. It achieves this by these simple questions:

**Table 1. Scrum basics**

| *Scrum asks…* | *Fundamental Project Management issue* |
|---|---|
| What have you done during the last 24 hours? | This is progress, it's work completed to date |
| What do you plan to do in the next 24 hours? | This is forward planning, it is work you are about to do |
| What's stopping you getting on with the work of the next 24 hours? | These are your impediments or obstructions, it might be things you need in order to work… more forward planning. It's also identification of immediate risks. |

### 2.2.2 Why Scrum

Scrum is a lightweight agile process framework used primarily for managing software development. Scrum clearly provides many benefits for the developers, but what are its advantages for the client and other stakeholders? Benefits for the client (Schwaber, 2010):

**Scrum's main advantage** is the way it facilitates changing customer requirements. It does this by providing a flexible framework wherein new features can be added and re-prioritizations can take place without negatively affecting the project flow and team's morale.

**Secondly,** the system of sprints encourages regular feedback, as each sprint is followed by a review. These reviews provide opportunities for the entire team to reflect on the previous sprint and discuss improvements for the next.

**Thirdly,** by defining roles for team members it promotes collaboration as well as clear and open lines of communication between the developers, the client and other stakeholders.

**Finally,** Scrum provides a framework for work estimation, where features are estimated in units of points: each point represents a relative amount of effort required. This technique is flexible enough to allow for changes in requirements as well as changes in the development team's velocity.

However, Scrum is not suitable for all projects and has certain limitations:

- It does not work with traditional software development methodologies such as Waterfall.
- It cannot be fully adopted if a team is too small or too big – an ideal team would consist of 5 to 10 developers. It does not work well with huge groups or with people who have significant responsibilities elsewhere.
- It cannot work effectively unless it has full management / client support.
- It does not work well if team is unable to define goals, or break them down into small, quantifiable chunks.
- It requires a Scrum Master who understands the Scrum practices and is able to apply them.

## 2.3 Summary

Agile software development is a group of software development methodologies based on iterative and incremental development models.

Agile assumes that a customer is usually unable to explain all the requirements before starting the project and due to close and frequent collaboration of a client and development team, the requirements, as well as changes in requirements are discussed, reviewed and implemented in each iteration cycle. Since, the requirements are changing dynamically, one of the most important characteristics of the agile software development methodology becomes the development team ability to respond quickly and efficiently to those changes.

Agile is currently one of the most popular software development methodologies. Some advantages that agile development represents are:

- Agile methodology has an adaptive team which is able to respond to the changing requirements.
- The team does not have to invest time and effort and finally find that by the time they delivered the product, the requirement of the customer has changed.
- Face to face communication and continuous inputs from customer representative leaves no space for guesswork.
- The documentation is crisp and to the point to save time.
- The end result is the high quality software in least possible time duration and satisfied customer.

# 3 User Story

User stories describe the benefits of products delivered by the project in a simple format. Many Scrum teams have adopted the user story template developed by Mike Cohn, which identifies who the end user is, what the end user wants, and why in a single sentence. This model of the user story is most often written like this: "As a [end user role], I want [the desire] so that [the rationale]. In summary, user stories document requirements with particular attention to the end user's point of view.

- As a **user** (i.e. visitor of the website)
- I want functionality / **goal** (i.e. submit my email address)
- In order to / what **reason** (i.e. I can receive the newsletter)

To have such format for the user story is very useful and could be explained within some reasons:

- Obviously by saying "As a such – and – such, I want…." you can see how the person's mind goes instantly to imagining he or she is a such – and – such.
- Having a structure to the stories actually helps the product owner prioritize. If the product backlog is a jumble of things like Fix exception handing, Let users make reservations, Users want to see photos, Show room size options, and so on, the product owner has to work harder to understand what the feature is, who benefits from it, and what the value of it is (Cohn, 2004).

## 3.1 User Stories, Epics and Themes

### 3.1.1 Agile Themes

A Theme is a top – level objective that may span projects and products. Themes may be broken down into sub – themes, which are more likely to be product – specific. At its most granular form, a Theme may be an Epic.

Themes can be used at both Program and Project Level to drive strategic alignment and communicate a clear direction.

### 3.1.2 Agile Epics

An Agile Epic is a group of related User Stories. You would be unlikely to introduce an Epic into a sprint without first breaking it down into its component User Stories so to reduce uncertainty.

Epics can also be used at a both Program and Project Level – Read more using Epic Boards to manage programs and projects.

### 3.1.3 Agile User Stories

A User story is an Independent, Negotiable, Valuable, Estimable, Small, Testable requirement ("INVEST Acronym"). Despite being Independent i.e. they have no direct dependencies with other requirements, User stories may be clustered into Epics when represented on a Product Roadmap (Wake, 2003).

User Stories are great for Development Teams and Product Managers as they are easy to understand, discuss and prioritize – they are more commonly used at Sprint – level. User Stories will often be broken down into Tasks during the Sprint Planning Process – that is unless the stories are small enough to consume on their own.

## 3.2 Format of User Story

User stories have three critical aspects. We can call these Card, Conversation, and Confirmation.

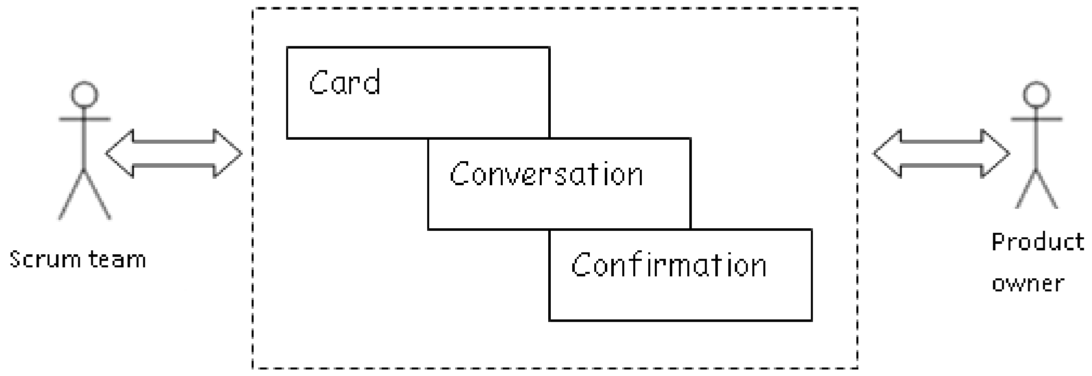


**Figure 3. User story workflow**

In all three aspects either scrum team and product owner are much closed related in collaboration through user story format that allows clarifying user stories and features to the end user (Jeffries, 2009).

### 3.2.1 Card

Usually user stories are written on cards. The card does not contain all the information that makes up the requirement. Instead, the card has just enough text to identify the requirement, and to remind everyone what the story is. The card is a token representing the requirement. It's used in planning. Also all needed notes are written on it, reflecting priority and cost. It's often handed to the programmers when the story is scheduled to be implemented, and given back to the customer when the story is complete.

### 3.2.2 Conversation

The requirement itself is communicated from customer to programmers through conversation: an exchange of thoughts, opinions, and feelings. This conversation takes place over time, particularly when the story is estimated (usually during release planning), and again at the iteration planning meeting when the story is scheduled for implementation. The conversation is largely verbal, but can be supplemented with documents. The best supplements are examples; the best examples are executable, we could call these examples confirmation. The Conversation section should provide more information about the feature.

### 3.2.3 Confirmation

No matter how much discussion or how much documentation we produce, we cannot be as certain as we need to be about what is to be done. The confirmation in the user story's key aspects adds confirmation that we sorely need. This component is the acceptance test (Jeffries, 2009).

At the beginning of the iteration, the customer communicates to the team manager / scrum master what s/he wants, by telling them how s/he will confirm that they've done what is needed. S/he defines the acceptance tests that will be used to show that the story has been implemented correctly.

At the end of the iteration, when the story is done, the programmers show the customer that the story is done, confirming success by showing that the acceptance tests run correctly.

The confirmation provided by the acceptance test is what makes possible the simple approach of card and conversation. When the conversation about a card gets down to the details of the acceptance test, the customer and programmer settle the final details of what needs to be done. At the end of each iteration the scrum team demonstrates that acceptance tests run successfully.

## 3.3 Requirements for the User story

First thing that we are facing in development is requirements. Having the right requirements I see the most painful point. Having a good story can change how the requirement is delivered and tested. According to Bill Wake a story should have the following features INVEST (Wake, 2003).

- I – Independent
- N – Negotiable
- V – Valuable
- E – Estimable
- S – Small
- T – Testable

Most of them are self – explanatory, but I will explain each of them in more detail.

### 3.3.1 Independent

One user story should be independent of another (as much as possible). Dependencies between stories make planning, prioritization, and estimation much more difficult. Often enough, dependencies can be reduced by either combining stories into one or by splitting the stories differently. Stories are easiest to work with if they are independent.

If the user stories are tightly dependent, a good idea might be to combine them into a single user story.

The main question to evaluate independency of the user story is:

- Is this user story is self – contained (in a way that there is no inherent dependency on another user story)?

### 3.3.2 Negotiable

A user story has to be negotiable. The "Card" of the story is just a short description of the story which does not include details. The details are worked out during the "Conversation" phase. A "Card" with too much detail on it actually limits conversation with the customer. A good story captures the essence, not the details (Wake, 2003).

The main question to evaluate negotiability of the user story is:

- Can the user story always be changed and rewritten as that is a part of the sprint?

### 3.3.3 Valuable

Each story has to be of value to the customer (either the user or the purchaser, we do not care about value to just anybody; it needs to be valuable to the customer). A story needs to be valuable. The focus here is to bring actual project-related value to the end-user. Coming up with technical stories that are really fun to code but bring no value to the end-user beats one of the Agile Principles One very good way of making stories valuable is to get the customer to write them (Wake, 2003).

The main question to evaluate valuably of the user story is:

- Is this user story delivering a value to the end user?

### 3.3.4 Estimable

The developers need to be able to estimate a user story to allow prioritization and planning of the story. Problems that can keep developers from estimating a story are: lack of domain knowledge (in which case there is a need for more Negotiation / Conversation); or if the story is too big (in which case the story needs to be broken down into smaller stories).

A good story has to be estimated. If a user story size cannot be estimated, it will never be planned, tasked, and, thus, become part of a Sprint We do not need an exact estimate, but just enough to help the customer rank and schedule the story's implementation. Being estimable is partly a function of being negotiated, as it's hard to estimate a story we do not understand. It is also a function of size: bigger stories are harder to estimate. Finally, it's a function of the team: what's easy to estimate will vary depending on the team's experience. (Sometimes a team may have to split a story into a (time – boxed) "spike" that will give the team enough information to make a decent estimate, and the rest of the story that will actually implement the desired feature.)

The main question to evaluate estimatability of the user story is:

- Is the user story estimable?

### 3.3.5 Small

A good story should be small in effort, typically representing no more, than 1 – 2 person weeks of effort (some teams restrict them to a few person-days of work), or try to keep your user story sizes between 1 – 16 story points. Story point is an evaluation unit of the user story. User stories should not be so big as to become impossible to plan / task / prioritize with a certain level of certainty. A story which is more than that in effort can be too hard to know what‘s in the story‘s scope, can have more errors associated with scoping and estimation (smaller stories tend to get more accurate estimates).

Alistair Cockburn described the cards as tokens promising a future conversation. The details can be elaborated through conversations with the customer.

The main question to evaluate size of the user story is:

- Is this story small (Can the user story be sized in the range of 1 – 16)?

### 3.3.6 Testable

A story needs to be testable for "Confirmation" to take place. We do not develop what we cannot test. If we cannot test it then we will never know when we are done. An example of a non – testable story: "*software should be easy to use*". We should always keep in mind that a story should be considered DONE, among other things, if it was tested successfully. If a user story cannot be tested due to lack of information, the story should not be considered a good user story. This is very important for teams employing TDD – Test Driven Development. The user story or its related description must provide the necessary information to make test driven development possible.

The main question to evaluate testability of the user story is:

- Can the user story be tested and considered as DONE?

### 3.3.7 Evaluation INVEST model

Evaluation of the INVEST model is very important for the evaluation user story in total. To do it, we have to evaluate each unit / feature of the INVEST model from range 1 to 5, 1 – lowest, 5 – highest.

**Table 2. INVEST model evaluation and metrics**

| *Features* | *Evaluation question* | *Evaluation* |
|---|---|---|
| Independent | Is this user story is self – contained (in a way that there is no inherent dependency on another user story)? | [1…5] |

| | | |
|---|---|---|
| Negotiable | Can user story always be changed and rewritten as that is a part of the sprint? | [1…5] |
| Valuable | Is this user story delivering a value to the end user? | [1…5] |
| Estimable | Is the user story estimable? | [1…5] |
| Small | Is this story is small (is user story can be sized in range of 1 – 16)? | [1…5] |
| Testable | Can user story be tested and considered as DONE? | [1…5] |

Questions are structured that possible answers will be simply and clearly understandable.

- No – 1;
- Maybe no – 2;
- Maybe – 3;
- Maybe yes – 4;
- Yes – 5;

Evaluation of the user story based on the INVEST model has to be implemented by all participants in Scrum. For example the product owner or stakeholders evaluate the value of the user story, because most likely, they know the best value of each feature for the end user. The scrum team including QA, testers, designers and scrum master evaluate the rest of the features in the INVEST model. Testers of course should concentrate on the testability of the user story.

**Table 3. Evaluation example for INVEST model**

| *User Story* | *Independent* | *Negotiable* | *Valuable* | *Estimable* | *Small* | *Testable* | *Total* |
|---|---|---|---|---|---|---|---|
| ID 1 | 2 | 4 | 2 | 1 | 4 | 5 | **18** |
| ID 2 | 4 | 2 | 3 | 3 | 2 | 4 | **18** |
| ID 3 | 1 | 1 | 3 | 2 | 2 | 4 | **13** |
| ID 4 | 5 | 4 | 4 | 4 | 5 | 5 | **27** |
| ID 5 | 2 | 1 | 1 | 1 | 2 | 2 | **9** |
| ID 6 | 1 | 1 | 5 | 1 | 1 | 1 | **10** |
| ID 7 | 5 | 5 | 5 | 5 | 5 | 5 | **30** |
| ID 8 | 3 | 5 | 3 | 4 | 3 | 5 | **23** |

Example of the evaluation for the INVEST model by user stories. Usage of the INVEST model evaluation in the calculation of the changeability rate will be introduced later.

## 3.4 Prioritizing user stories

Prioritizing user stories is one of the most important tasks for the team leader (i.e. scrum master). Because we have to have working code that represents a business value to the customer after each iteration, we have very carefully prioritized user stories. Some questions that we have to be able to answer before prioritizing (Soldi, 2011):

**Table 4. Metrics for the prioritizing user stories**

| *Metrics* | *Answer* |
|---|---|
| Is it really important that these stakeholders (users) are able to do this? | YES / NO |
| Is it actually possible for us to support this activity currently? | YES / NO |
| Is it important enough to us that we should consider infrastructure / policy changes? | YES / NO |
| Can we meet these goals only in this way? | YES / NO |
| Do we need to meet these goals now? | YES / NO |
| Is this a long term project goal? | YES / NO |

If we can answer YES to more than three answers then we should implement the user story otherwise we should re – think / re – design, suspend to the next iteration or even remove the story from the product backlog and consider it as inaccurate a user story.

To prioritize user stories we have use the same factors as in calculating changeability rate. Risk and Value (business value for the customer). Firstly, we have to evaluate every user story from value perspective. This evaluation of value for the user story has to be done by product owner or stakeholder. Of course they have to know what is needed and which features are necessary for the end user. Higher values have to be assigned to each activity that directly has influence to one or more objectives to be achieved. Lower values have to be assigned to activities that not so important in achieving the goals such as maintenance, administrative, reports, etc. This evaluation has to be done using indexes from 1 to 10. 1 has the lowest value index, 10 has the highest value index.

After the value evaluation, we have to evaluate the user story from the risk perspective. This evaluation has to be done by the project manager / scrum master. Higher risk has to be assigned to critical features, features that need to be implemented by applying new technologies, new methodologies that are not well described, those that have many interactions with external systems, different environments those that will serve different typology of users, etc. Low risks have to be assigned to the remaining features. This evaluation also has to be done using indexes from 1 to 10. The lowest index means the lowest risk and vice versa.

Secondly, we need to put all risk and value evaluations of our user stories into a table

**Table 5. Prioritizing user stories by risk and value**

| *User Story ID* | *Risk* | *Value* |
|---|---|---|
| User story 1 | 2 | 3 |
| User story 2 | 5 | 2 |
| User story 3 | 8 | 8 |
| User story 4 | 7 | 6 |
| User story 5 | 9 | 4 |
| User story 6 | 3 | 9 |
| User story 7 | 1 | 7 |
| User story 8 | 2 | 5 |

Now we can make a graph based on the data derived from our evaluations (see table). The figure below shows us how we can clarify our results by displaying the data as points in a graph divided into four main areas.

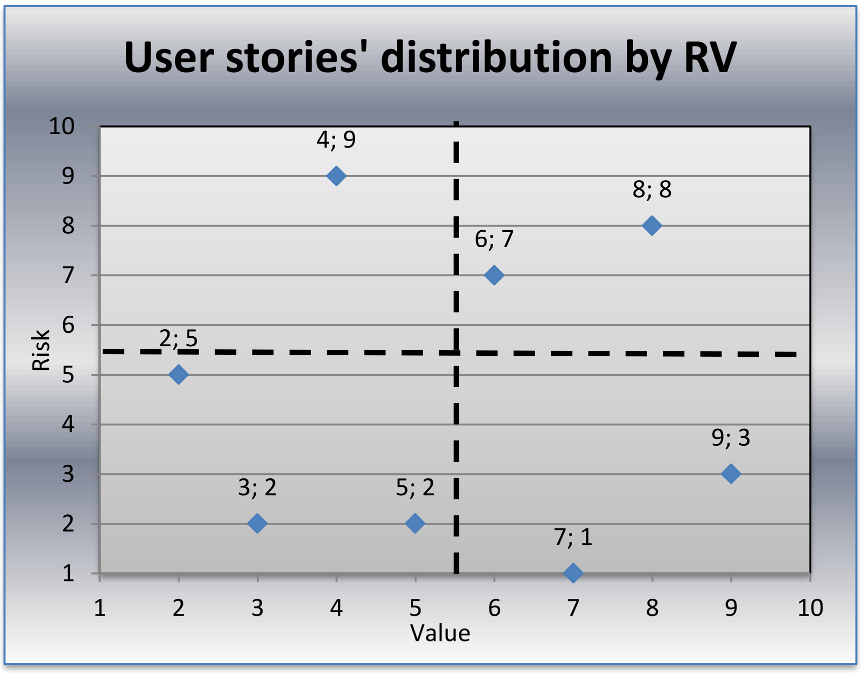


**Figure 4. User stories' distribution by risk and value**

The following figure shows us how priority depends on risk and value, higher value and higher risk gives the user story high priority, low value and low gives the user story low priority, respectively the user story with medium priority is defined with high value and low risk. The described sub quadrants are quite clear and simple enough evaluate the user story by risk and value, dependent in which sub quadrant it belongs to, providing a clear priority value for the user story. The most important sub quadrant for us is the one that identifies the user story with low value and high risk. Our main questions are:

- What to do with the user story that falls into such sub quadrant?

- Do we have to remove such user story form our product backlog?

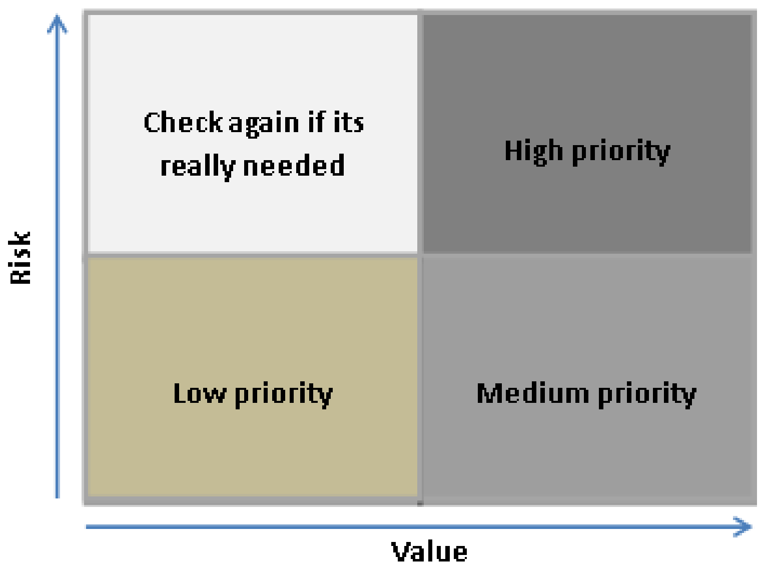


**Figure 5. Quadrants for prioritizing user stories**

Based on the graph above, we can identify the user stories that we have to review for the implementation. From the graph based on the data from table 5 and from the graph's defined quadrants for prioritizing user stories we can see that user story #5, with Value of 4 and Risk of 9, falls into a sub quadrant that requires us to re – think / re – design the user story. In this case we have to review the user story using the methods described earlier. If our re – evaluation provides us with a lower risk rating or a higher value rating than earlier than that user story should be implemented otherwise try to re – think / re – design, maybe then that user story will falls into a different sub quadrant and can be prioritized. On the other hand we can always suspend a user story till the next iteration.

## 3.5 Changeability of the User Story

Two main factors have to be taken into account then evaluating changeability. These factors are Risk and Value. Every user story has to be assigned to one of the following categories:

- High risk – low value
- High risk – high value
- Low risk – low value
- Low risk – high value

First of all what are risk and value?

- Risk – one thing to consider is the scrum team's experience level.
- Value – implemented story's importance to the customer.

Assigning RV (risk & value) numbers onto issues we can assume a changeability rate in the following manner:

1. If we have a user story with high risk and low value evaluations, it might mean that team is inexperienced. Where the value for the customer is low, the changeability rate should be minimized.

2. If we have a user story with both high risk and high value evaluations it might mean that team is not experienced. However the value for the customer is high this story might require a lot of rework in the future. In this case the changeability rate should be maximized.

3. If we have a user story with low risk and low value evaluations the development of such user story should be quite predictable. The team's experience and the low value to customer decrease the likelihood of changes in the future. So, the changeability rate of such a user story is quite low.

4. If we have a user story with low risk and high value evaluations it might mean that the development team's experience decreases the risk of rework, but the high value for the customer might increase the need for future changes. So, the changeability rate should be somewhere between case 2 and 1.

Prioritizing these cases, according to changeability rate gives us the following order: case 2, case 4, case 1, and case 3.

Looking at the story's changeability rate, we only concentrate on future changes of the requirements from the customer. Changes in the testing phase are not included. Any fixes required after testing phases would require us to predict the defect rate of each user story, which is a completely different challenge.

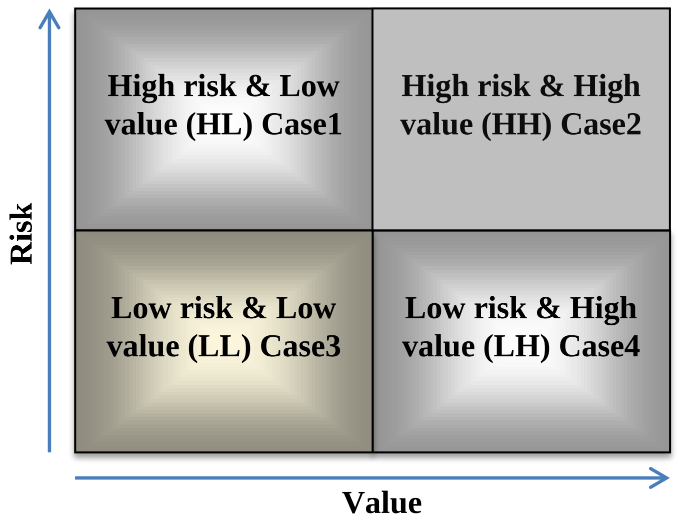

High risk & Low value (HL) Case1
High risk & High value (HH) Case2
Low risk & Low value (LL) Case3
Low risk & High value (LH) Case4
Risk
Value

**Figure 6. Changeability quadrants by risk and value**

This changeability quadrant works only for risk and value. If we only use these variables, we can depend on this evaluation. However, if we add evaluation of our INVEST model our changeability rates change significantly.

Let's use our previous data with evaluations of the points for the user story by risk and business value and the total evaluation of the INVEST model.

**Table 6. Example values for the changeability rate**

| ***User Story ID*** | ***Risk*** | ***Value*** | ***INVEST model*** |
|---|---|---|---|
| User story 1 | 2 | 3 | 18 |
| User story 2 | 5 | 2 | 18 |
| User story 3 | 8 | 8 | 13 |
| User story 4 | 7 | 6 | 27 |
| User story 5 | 9 | 4 | 9 |
| User story 6 | 3 | 9 | 10 |
| User story 7 | 1 | 7 | 30 |
| User story 8 | 2 | 5 | 23 |

Let's look closer at these equations:

$$ChR = \left(\frac{R}{R_{Max}} + \frac{V}{V_{Max}} + \left(\frac{INVEST_{Max} - INVEST}{INVEST_{Max}}\right)\right) \times 0.25 \tag{1}$$

$$ChR' = \left(\left(\frac{R}{R_{Max}} + \frac{V}{V_{Max}}\right) \div 2\right) \times 0.25 + \left(\frac{INVEST_{Max} - INVEST}{INVEST_{Max}}\right) \tag{2}$$

*Where:*

$ChR$ – Changeability rate;

$ChR'$ – Changeability rate;

$R$ – Risk defined of user story;

$R_{Max}$ – Maximum possible value of the risk, in our case always 10;

$V$ – Value defined of user story;

$V_{Max}$ – Maximum possible value of the value, in our case always 10;

$INVEST$ – Defined evaluation of the INVEST model for the user story;

$INVEST_{Max}$ – Maximum possible value of the evaluation of the INVEST model, in our case always 30 (6 units x 5);

0.25 – Constant represents sub quadrant absolute value.

Natural question:

- Why do we have two equations for calculating the changeability rate of the user story?
- What is the main difference between these equations?
- When should we use one or other?

Answers to these questions are quite simple. Firstly, if we look closer at these equations we realize that the influence of the evaluation of the INVEST model in the second equation is higher than in the first one. So, if we rely on the evaluation of the INVEST model of the user story we should choose the second one. Secondly, every agile project has different complexity and these equations depend on the whole complexity of the project. If the project's complexity is evaluated as medium or high, we should also use equation number 2, because such projects usually have longer iterations and the INVEST model has to be more accurate. Equation number 1 we should use with smaller and lighter projects, f.e. mobile app projects. On the other hand we could use different equations in the same project in different iterations. For more details see Appendix – Case Study.

**Table 7. Changeability rate**

| ***User Story ID*** | ***Risk*** | ***Value*** | ***INVEST model*** | ***Changeability Rate [1]*** | ***Changeability Rate [2]*** |
|---|---|---|---|---|---|
| User story 1 | 2 | 3 | 18 | 0,23 | 0,46 |
| User story 2 | 5 | 2 | 18 | 0,28 | 0,49 |
| User story 3 | 8 | 8 | 13 | 0,54 | 0,77 |
| User story 4 | 7 | 6 | 27 | 0,35 | 0,26 |
| User story 5 | 9 | 4 | 9 | 0,50 | 0,86 |
| User story 6 | 3 | 9 | 10 | 0,47 | 0,82 |
| User story 7 | 1 | 7 | 30 | 0,20 | 0,10 |
| User story 8 | 2 | 5 | 23 | 0,23 | 0,32 |

Then we define the changeability rate we can not only depend on the equations perspective, we have to keep in mind the changeability quadrant aspect. We have to identify which sub quadrant our user story falls into and then compare those results with the results of the selected equation. There is no one rule to define the changeability rate. We have to combine our selection based on all conditions.

Because Agile allows changing requirements at any time in iteration we have to be ready to alter changeability rate. Some issues:

- Changeability rate can be changed in the next iteration, based on team experience;
- Changeability rate can be changed or not used based on similar user stories in similar projects.

## 3.6 Procedures and Work Instructions

A *procedure* is a particular way of accomplishing something or of acting. *Work instructions* are used mainly in cases where a uniform method of performing the task throughout the organization is either impossible or undesirable. In our case, as a result, work instructions are specific to an agile team.

Procedures and work instructions aim at (Galin, 2004):

- The performance of tasks, processes or activities in the most effective and efficient way without deviating from quality requirements. This implies that procedures need to be reviewed from time-to-time and not left to drift into obsolescence.
- Effective and efficient communication between the separate employees involved in the development and maintenance of software systems. Uniformity in performance, achieved by conformity with procedures and work instructions, reduces the misunderstandings that lead to software errors.
- Simplified coordination between tasks and activities performed by the various bodies of the organization. Better coordination means fewer errors.

Simply:

- Procedures – tell you **who** does **what when**.
- Work Instructions – tell you **how** to do something.

Remember: "**who**" means title or department not actual name. "**When**" generally means in what sequence, not clock time.

### 3.6.1 Procedures

A procedure is a way or manner of doing something that usually involves a series of calculated steps. Procedures are the detailed activities or processes to be performed according to a given method for the purpose of performing a certain task.

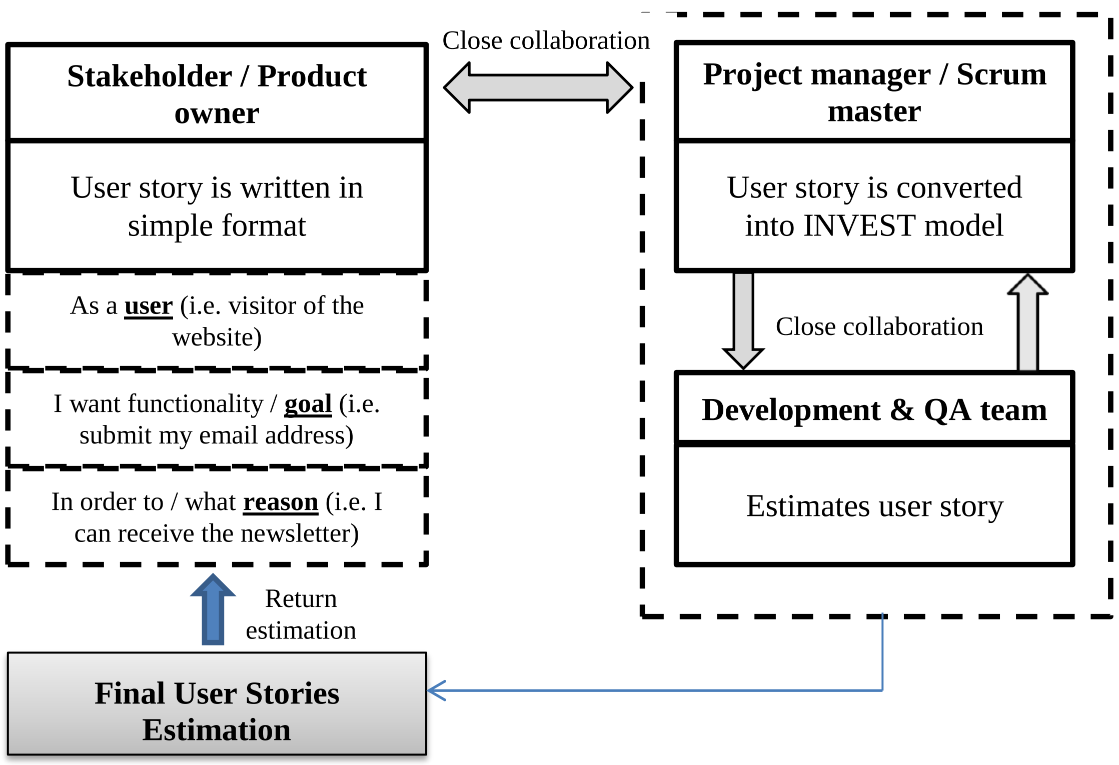

Stakeholder / Product owner
User story is written in simple format
As a user (i.e. visitor of the website)
I want functionality / goal (i.e. submit my email address)
In order to / what reason (i.e. I can receive the newsletter)
Close collaboration
Project manager / Scrum master
User story is converted into INVEST model
Close collaboration
Development & QA team
Estimates user story
Return estimation
Final User Stories Estimation

**Figure 7. User story's definition and estimation workflow**

Stakeholder or Product Owner is responsible for writing user stories in simple format. In the beginning of the project, the project manager provides a simple format for the user stories to the product owner. It is his / her responsibility to write user stories in such format in close collaboration with the project manager or the scrum master. The product owner provides all user stories with evaluated business value to the customer, for at least the first release of the project, to the scrum master. The scrum master converts all these user stories into an INVEST model. Finally the product owner and the scrum master prioritize user stories. At this point the collaboration with the product owner is finished. After the user story is ready the scrum master with development and QA team (note: developers refer to all programmers, testers, database engineers, analysts, user interaction designers, and so on.) evaluates risk and the INVEST model of the user story and estimates it through a planning poker session. Usually estimating a user story takes 2 – 3 rounds of a planning poker session.

After the user story is estimated and prioritized the scrum master calculates the changeability rate of each user story and adds it into final estimation of the user story.

The scrum master returns the final user stories estimation to the product owner for the release divided by iterations if the project release plan consists of more than one iteration.

### 3.6.2 The Five W’s

Procedures supply all the details needed to carry out a user story. These details can be viewed as responding to five issues known as the Fives W´s (Galin, 2004):

- **W**hat activities have to be performed?
- Ho**W** should each activity be performed?
- **W**hen should the activity be performed?
- **W**here the activity should be performed?
- **W**ho should perform the activity?

Let’s talk about each issue in more detail.

Firstly by procedure we have to write a user story, based on requirements from the customer

**What?** In the beginning the user stories have to be written into simple format. This is the easiest way to have the requirements on the card. Then each user story has to be reviewed from 3C’ concept:

- Card
- Conversation
- Confirmation

**HoW?** Every user story has to be written / reviewed in close collaboration between Stakeholders / product owner and project manager / scrum master. The product owner evaluates a value for the user story and the scrum master evaluates risk of it. This is an important issue, keep in mind that the value and risk of the user story cannot be evaluated only by one participant it has to be done within the collaboration.

**When?** One of the main principles of agile methods is to “welcome changing requirements” (Beck et al., 2001), however changing requirements are a major issue. Thus an agile team is ready to accept changing requirements in any phase of the iteration, either in the beginning or in the end.

**Where?** This issue does not apply to the user story. Importance of location is not necessary for the defining a user story. The process could be done anywhere you like.

**Who?** Stakeholders write user stories (in format as a [end user role], I want [the desire] so that [the rationale]). An important concept is that your project stakeholders write the user stories, not the developers. User stories are simple enough that people can learn to write them in a few minutes, so it makes sense that the domain experts (the stakeholders) write them. After that the project manager takes these “user stories” and rewrites them INVEST model form in close collaboration with the stakeholders or product owner.

### 3.6.3 Meeting Work Instructions

Work instructions are a set of instructions for performing a task or for following a procedure. Work instructions are used where a uniform method is either impossible or undesirable. Work instructions are specific to a team or department. They supplement procedures by providing explicit details what are suitable solely to the needs of one team, department or unit (Galin, 2004).

**Table 8. Meeting work instructions**

| # | *Steps* | *Key points* | *Reasons for key points* |
|---|---|---|---|
| 1. | Schedule Meeting | - Make sure to invite all stakeholders;<br>- Schedule way ahead;<br>- Use calendar invitations;<br>- Make sure to confirm by phone / email. | - The User story meeting is the best moment to crop all different views, requirements and interests<br>- So the maximum amount of stakeholders can actually be present<br>- So meetings are scheduled in the right calendars<br>- To get a spoken confirmation of all participants and also because not all stakeholders use the same calendar |
| 2. | Prepare Meeting | - Use existing documentation to list the required themes (features) and user stories<br>- Make sure to translate the documentation into known user stories<br>- Make sure to use common knowledge of themes and user stories | - You do not want to redo the entire request for the project<br>- You want to prefab the user stories needed, in order not to lose time on obvious thing during the meeting<br>- In order to move faster this time and use experience gained |
| 3. | Prepare Meeting Room | - Make sure to have clear walls<br>- Make sure to have all materials ready: Markers for everyone, sufficient index cards, pins and/ or tape<br>- Write out the existing User stories on index cards and pin these on the wall | - In order to fill these with User stories<br>- In order to be able to work efficiently during the meeting<br>- So everyone notices that time is properly made use of |

| | | | |
|---|---|---|---|
| 4. | Start User Story Meeting | - Explain the purpose and method of the meeting<br>- Provide a clear agenda for the meeting: Project brief, User stories, categorization and time frame<br>- Make sure everyone is properly introduced<br>- Be brief, but not too brief (15 minutes is ideal) | - So everyone knows the goal and meaning of the exercise<br>- So everyone knows what activities to expect<br>- So everyone can easily address one another<br>- So the meeting can start quickly and everyone has time to arrive and accommodate |
| 5. | Project Brief | - Provide an introduction / description of the project<br>- Preferably, let the client or commissioner provide the brief<br>- Keep it brief (15 minutes max.) | - So everyone has a shared vision of what needs doing<br>- So the entire teams hears the first version, rather than an interpretation and also for the project owner to pick up his role<br>- This is not the moment to dive into every possible detail of the project |
| 6. | User Story Recap | - Read out the existing user stories to the team | - So everyone has a clear understanding of what shall be done |
| 7. | User Story Completion | - Ask the team to state additional user stories<br>- Ask the team to fine tune existing user stories<br>- Take a positive attitude, accept all user stories at first<br>- Allow divergence first<br>- Use creativity techniques | - In order to have a complete view of the entire project<br>- In order to maximize team knowledge and be as accurate as possible<br>- In order to promote team member participation<br>- In order to allow unexpected ideas to pop up<br>- In order to enrich the project as much as you can |
| 8. | User Story Categorization | - Have the team organize the user stories by theme<br>- Try to combine some stories and to reduce the total amount of stories<br>- Try to split complex stories in several simple stories<br>- Write the theme on the | - In order to promote team ownership and team member inclusion<br>- In order to avoid double work<br>- In order to come up with a list of actually doable user stories<br>- In order to easily recognize the theme to which the story |

| | | | |
|---|---|---|---|
| | | upper left corner of the card | belongs |
| 9. | Conclude User Story Meeting | - Allow for final questions and remarks<br>- Thank everybody for their participation<br>- Communicate the next steps | - In order to make sure that everything is said<br>- In order to appreciate all effort put into the exercise<br>- So everyone is on the same page |

User story meetings are preferably done with clients and if possible with a couple of end users who are able to provide direct input on what they want and what for. The goal of the User story meeting is to get a maximum of insight into all different requirements for the project, in a minimum of time. A second goal of the User story meeting is to ensure that all stakeholders get a full understanding of the scope of the project -and also of perhaps conflicting requirements. User story meetings are the most efficient way to get a maximum of shared insight into all different requirements for the project.

## 3.7 Recognize Bad user story

We already talked about how we should write user stories. Let's take a look how to recognize bad user stories. To recognize them it's project manager's responsibility (Cohn, 2004).

1- <u>User story are only a wrapper</u>

If user stories only consist of one task, this is a sign that you just using them as a wrapper. A user story is "a promise for a conversation". If there is nothing to discuss and it is a simple task, than you have to rethink it. Also it's a sign that user story is too small.

2- <u>User story can be done in less than a day</u>

If you have 40+ user stories on your Sprint Backlog, it gets really hard to get a commitment from your team. From the INVEST model we know, that we have to write small user stories, but definitely not too small. A small user story is great, but not if all of these stories are tasks. In such cases, these stories belong to a bigger story and do not make sense on their own. Check if your stories are meant to add new functionality, if not something is wrong.

3- <u>User story does not describe a feature</u>

This is somewhat related to 1 and 2, because in most cases these signs come in together. If user story is describing tasks for the developer, instead of describing a new functionality or feature for the customer, something went wrong.

4- <u>User story focus on wrong user</u>

User stories written like: "As a project manager…", "As a product manager….", "As a developer…" or "As a test technician…"? Than you have lost the real user of your software. It will not be the project manager or product manager who will use your software. In this case start over and ask yourself who will use it and write your user stories with their view in your mind. Another possibility would be to create personas. But always have your user in the focus.

5- Platinum plate

Resist the urge. It takes time to implement, test and document things that the user did not ask for and may not need. By agile manifesto: “we develop only what we ask for”. For example:

- **User story:** A scientist can view the instrument data in a landscape plot that is scrollable and scalable.
- **The platinum plating would be:** A scientist can view the instrument data in a plot whose aspect is configurable that is scrollable and scalable and zoomable and may change color.

Do not try to give user more than they ask for.

6- Customer cannot prioritize

The customer needs to be able to prioritize the user story. It means that feature has to be valuable and to have value to the customer. Invaluable user story cannot be prioritized!

7- Thinking too far ahead

Its classic! This mistake is very common while defining the user story. It’s highly not recommended to think about the feature too far. It leads to prioritize issue such stories are more difficult to prioritize. Also usually such stories are too big.

8- User story too big

Usually this mistake relates with “thinking too far ahead” mistake. Often this issue rises in the non – experienced teams. There are some identifying signs of the being user story too big:

- Not estimated to fit in one iteration;
- Team is repeatedly moving partial stories out of iteration into the next one.

## 3.8 Tools

There exist plenty of various tools for aiding agile developers. Basically, most tools for managing user stories are the part of the whole agile project management tool. Most of them have the possibility to manage user stories in terms of capturing and tracking stories that are currently being developed.

In the following paragraph some of the most common tools are introduced.

**Table 9. Agile project management tools**

| *Software* | *Application type* | *License* | *Last updated* |
|---|---|---|---|
| GreenHopper (for JIRA) | Browser | Commercial, Free | July 2010 |
| Mingle | Browser | Commercial | June 2011 |
| Scrumy | Browser | Commercial. Free | July 2010 |
| ProjectCards | IDE Plugin, Native | Commercial. Free | June 2010 |
| Trello | Browser | Free | February 2012 |

### 3.8.1 JIRA (GreenHopper plugin)

GreenHopper is a tool for managing your backlog, planning sprints and tracking your team through the entire release process.

License: Free (for nonprofit companies).

### 3.8.2 Mingle

Mingle is an Agile project management solution that offers visibility into software initiatives and collaboration among business and delivery teams. The only solution that’s truly Agile, Mingle lets teams work the way they work best, adapting as they evolve their processes, make better decisions and quickly respond to changes.

License: Commercial.

### 3.8.3 Scrumy

Scrumy is a project management tool loosely based off of Scrum. Scrumy has a free version that gives you user stories, todos and a virtual taskboard. With the free version, you do not get a backlog or protected access.

License: Free (for nonprofit companies).

### 3.8.4 ProjectCards

ProjectCards is project management tool. It allows entering and classifying requirements in the Themes view. Using drag & drop, can build Release and Iteration (Sprint) schedule in the Release Plan. Use the Project Dashboard to monitor the progress of your work. Define as many custom fields as you need to gather specific information about each card.

License: Free (for nonprofit companies).

### 3.8.5 Trello

Trello is a simple and powerful free web app project management tool. In its form developers describe it as a webpage where you make a bunch of lists easy to create. In a collaboration project or workflow giving the ability to keep track of progress. Using features such as adding lists as well as items (called cards) help in this. Members can add and edit items and lists as well as adding people to task by dragging frames around. In any card can be add all types of media, comments and checklists for tracking progress on that task.

License: Free

## 3.9 Summary

Agile software development relies heavily on a work item type called user stories. It’s very important for the whole agile projects and success of it because user stories represent features for the user.

A good user story describes the desired functionality, who wants it, and how and why the functionality will be used. The basic components of a User Story are sometimes dubbed as the three C's:

- Card – the written description of the story, serves as an identification, reminder, and also helps in planning.
- Conversation – this is the meat of the story; the dialogue that is carried out with the users; recorded notes; mockups; documents exchanged.
- Confirmation – the acceptance test criteria that the user will utilize to confirm that the story is completed.

Prioritizing user stories is one of most important task for the team leader (i.e. scrum master). Because we have to have working code that represents a business value to the customer after each iteration.

Changeability rate is another very important issue. Because Agile allows changing requirements at any time in iteration we have to be ready to count changeability rate. Some issues:

- Changeability rate can be changed in the next iteration, based on team experience;
- Changeability rate can be changed or not used based on similar user stories in similar projects.

Defining user stories close collaboration between stakeholder / product owner and scrum team is a key.

User stories has to be implemented into the INVEST model approach. Well written User Stories are cornerstones for Agile Development:

- They should be independent of each other;
- Their details should be negotiated between the users and the developers;
- The stories should be of value to the users;
- They should be clear enough for developers to be able to estimate them;
- They should be small;
- They should be testable through the use of pre-defined test cases.

Also badly defined / written story can lead a wrong estimation for the developer which can cause capacity issues for testing. Badly estimated stories could increase risk in further sprints.

Signs of the bad user story:

- User story are only a wrapper;
- User story can be done in less than a day;
- User story does not describe a feature;
- User story focus on wrong user;
- Platinum plate;
- Customer cannot prioritize;
- Thinking too far ahead;
- User story too big.

# 4 Velocity

*"It is better to be roughly right than precisely wrong."*–John Maynard Keynes

Velocity is an Agile (Scrum) term, and it is a metric to estimate the productivity of the project team based on their previous work such as previous iteration. Team velocity could be calculated as:

$$V_{t_1} = \frac{CSP}{I_{length}}$$

*Where:*

$V_{t_1}$ – Velocity of the Team ($t_1$ stands for team 1);

$CSP$ – Completed Story Points;

$I_{length}$ – Iteration length in days usually is 2 – 4 weeks (5 – 20 days) that took the team to finish this work.

**Table 10. Velocity by iteration**

| *Iteration* | *Iteration length* | *Completed Story Points (CSP)* | *Velocity* |
|---|---|---|---|
| 1 | 10 | 47 | 4.7 |
| 2 | 10 | 51 | 5.1 |
| 3 | 10 | 45 | 4.5 |
| 4 | 10 | 56 | 5.6 |
| 5 | 10 | 50 | 5 |
| 6 | 10 | 47 | 4.7 |
| **AVG** | | **49** | **4.9** |

For example, if the team in an iteration $n$ able to finish 50 story points and iteration length is 2 weeks (10 working days), then the team‘s velocity is 50 / 10 = 5 completed story points per day. Future iterations use the proven history of the team to determine how much the team can do. Therefore, velocity is the right measure to use for planning future iterations.

Velocity is specific to a team and comparisons to other teams are irrelevant. There are many reasons for this, but the main one is that different teams will develop different Estimates for the same task i.e. developers on Team 1 might rate the effort of a particular story at 6 while team 2 might rate it at 15. Further, teams have different skill sets and strengths that make them very effective in some areas, but not in others. Because of this, resist inter – team comparisons – they are simply not fair or accurate.

Velocity is measured in the same units as feature estimates, whether this is story points, days, ideal days, or hours – all of which are considered acceptable.

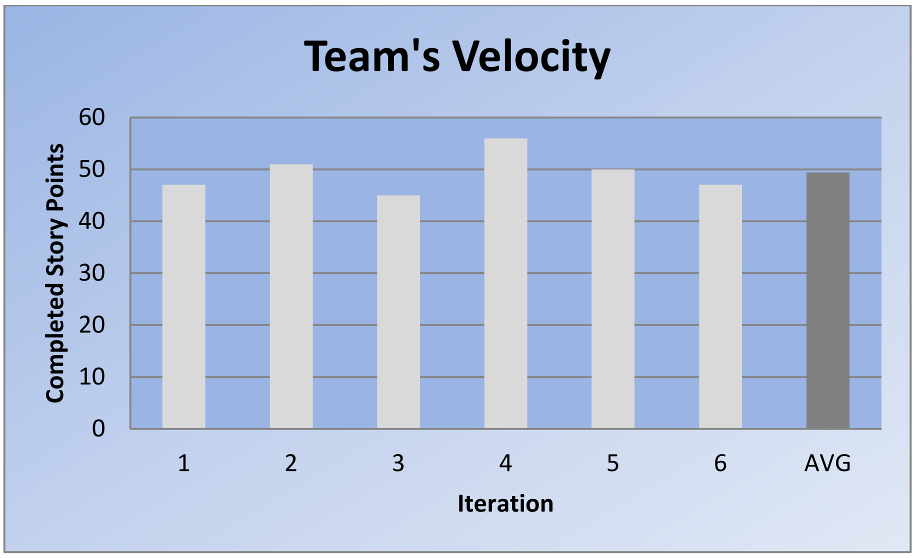


**Figure 8. Team's velocity**

From the team‘s velocity (figure 8), we see that CSP is fluctuating during iterations and affects velocity. Some iteration goes smoother than others, and story points are not always identical in terms of effort. Velocity will typically fluctuate within a reasonable range, which is perfectly fine. If velocity fluctuates widely for more than one or two iterations, the team may need to re-estimate and / or renegotiate the release plan. These fluctuations during iterations are natural at least in small ranges and are affected by resource fluctuations. For example If a productive team member retires or moves on to another opportunity, the team‘s velocity will almost certainly suffer. New hires usually cause a temporary drain on resources until they get up to speed team‘s overall velocity may go down for an iteration or two because experienced team members will spend time teaching the new stuff about the domain, development environment, team / company standards and values, and existing code. Reasons of fluctuation of the velocity could be very various (Lant, 2010):

- **Team changes:** Adding member, losing members, changing roles and responsibilities.
- **New tools:** Introduction of new development tools, database technologies, languages, etc… requires learning, and reduces Velocity until learned.
- **Vendor defects:** Defects in third party tools and software requiring developer workarounds eat into productivity and Velocity.
- **Responsibilities outside of the project:** Team members assuming additional responsibilities outside of the project. Shifting between projects can have a dramatic effect on productivity.
- **Personal issues:** Colicky baby at home, personal health, family dynamics, etc…
- **Stakeholders:** Stakeholders may not be responsive to requests for information from the developers or tester and thus create delays. They may also have unreasonable expectations of the team.

- **Unclear requirements:** Lack of clarity or detail in requirements cause unnecessary churns and rework.
- **Changing requirements:** New project specifications might require skills that are non-existent or weak in the team. Acquiring the skills, either by introducing new team members or by an existing team member acquiring the skills will impact productivity.
- **Relocation:** Moving the team to a new physical location upsets the rhythm and impacts their Velocity.

Velocity will quickly emerge during the first iteration. If underestimated, velocity in the first iteration will rise as new features are included; and if overestimated, velocity will decrease as features are removed. The second iteration should then use the first iteration as a guideline (Hearty et al., 2009).

- Do meetings, phone calls, email get included in velocity?

This depends on whether these items are estimated and included in the iteration plans. They are typically not included – a goal of velocity is relative consistency and predictability across iterations in terms of a team's ability to deliver. (Duarte, 2008)

- Should velocity be accumulated across teams or projects?

Velocity is very much a localized measure. In addition to different team members with different team 'personalities', projects typically possess unique characteristics in terms of estimating techniques, detail process, technology, customer involvement, etc. As a result, this can make organization-wide analysis very inaccurate. If, on the other hand, all of your teams estimate exactly the same, develop exactly the same, test exactly the same, and track exactly the same, then by all means, maybe you are the exception.

- How long does it take for velocity to stabilize?

In general team velocity will typically stabilize between 3 and 6 iterations

- Does maximum velocity mean maximum productivity?

Absolutely not. In an attempt to maximize velocity, a team may in fact achieve the opposite. If asked to maximize velocity, a team may skimp on unit or acceptance testing, reduces customer collaboration, skip fixing bugs, minimize refactoring, or many other key benefits of the various agile development practices. While potentially offering short – term improvement, there will be a negative long – term impact. The goal is not maximized velocity, but rather optimal velocity over time, which takes into account many factors including the quality of the end product. (Hearty et al., 2009)

- How do we measure velocity if our iteration lengths change?

Velocity's value comes from its inherent consistency. A fixed iteration length helps drive the reliable rhythm of a project. Without this rhythm, you are constant revising, re – estimating, and reconciling, and the ability to predict out in the future is minimized due to inconsistent results. If, almost everyone is going to be out a week for the holidays or a couple days for company – wide meetings, then by all means simply use common sense and adapt iteration dates or velocity accordingly. Like most agile practices, these are guidelines, not rules that are meant to prevent common sense. (Lant, 2010)

# 5 Estimating User Story

Story points come from extreme programming and propose measuring the workload in imaginable units of complexity. In general user story estimation is done in story points that are basic unit of the estimation. Also estimation could be done in ideal days or hours.

Advantages of the story point:

- Pure measure of size and complexity;
- Relative; longer shelf life;
- Independent of the estimator;
- Typically faster;
- Easier to work with;
- Studies show we are better at relative estimating.

Story Points are not a measurement of duration, but rather a measurement of size / complexity.

Team assumes one of the relatively simple tasks to be of size 1 or 2 and estimates the size of another task by comparing them to the chosen standard (Lant, 2010).

Pros:

- Estimations do not decay over time like ideal days do. When team skills go up, its velocity changes, not the work size;
- Relative values are what are important. It is easier to reach an agreement on relative sizes, than on a real world related artifacts like days;
- Pure measurements of work size – easy to sum.

Cons:

- More difficult to explain. The benefits of using unitless measurements and the power of relative estimations have to be explained

## 5.1 Prepare for the Estimation

*"In a good shoe, I wear a size six, but a seven feels so good, I buy a size eight."* – Dolly Parton in Steel Magnolia

### 5.1.1 Powers of two

Probably the most accurate way for the estimating user story is by comparison, because story points are relative values. One way to do it is to use power of two. Firstly we have to find the easiest user story that takes least effort to implement in our backlog. Let's assign it to 1. After having comparison starting point we can estimate rest user stories based on comparison. Choose another user story that is needed twice effort to implement than the first one, so, then it will be 2 story points. And then again, find another user stories that are needed twice effort to implement that the previous one and so on. Based on this way of the estimation we have story points i.e. 1, 2, 4, 8, 16, 32, etc.

Usually user stories should be estimated in range 1 – 16, if estimation of the user story is 32 or more, than we have to review it through the INVEST model perspective. Probably it's too big and it's needed to split it into smaller parts.

### 5.1.2 No averages or numbers not on the scale

Power of two values allows you to get a rough estimate without spending unnecessary time focusing on precision. Sometimes a story feels larger than an 8 but smaller than a 16. The story should not be estimated as a 10. There's really no reason to use a 10. The story carries enough risk or unknowns that it is not an 8; therefore, it's very likely that it will actually be a 16. Using an average or an off scale number can briefly (and unnecessarily) confuse a team member or stakeholder (Larman, 2003). Also, in the big picture of the project, the occasional uncommon estimate is not likely to make much of a difference. Keep it simple, stick to the scale.

### 5.1.3 Vote independently

It's human nature to be influenced by other people. If a technical leader says a story is a two, it's likely that the rest of the team will follow his lead. For this reason we should prefer an estimation process that lets each team member vote independently. This can be done by using sheets of paper that no one reveals until everyone is ready.

### 5.1.4 Take the largest estimate

Taking the largest estimate has additional benefits. If you must agree on a lower estimate then the team member with the larger estimate will need to discuss why they chose a larger value. This discussion can be uncomfortable for developers who are less senior on the team. They may not know how to do something as quickly, based on limited experience with the language or tools. Their concerns are often justified by their skill level, and it would be unfortunate if they felt uncomfortable giving their true estimate because they were afraid to discuss why it was higher. Any discussion around taking a higher or lower value may lead to the entire team raising their value, or it may lead to that developer uncomfortably lowering their estimate (Larman, 2003).

Finally, taking the largest estimate can help save time in an estimation meeting. If any member of the team believes the story is an 8 (s) he can speak up at any time while discussing the story and announce that (s) he is going to assign an eight. Unless someone else believes that there is a large estimation gap among team members, there's no reason to continue talking about the story since it will ultimately become an eight anyway.

### 5.1.5 Large estimate gaps

Large estimate gaps are common in estimating a user story. It's very rarely then the entire team agrees on the size of a story. However, sometimes a large gap represents a misunderstanding. For this reason any time there is a two value gap in estimation, additional conversation always occurs (e.g. if a team member assigns a 4 and another a 16, some clarification needs to occur). Discussing large gaps also ensures that taking the largest estimate has less chance of being abused (Moser, 2011).

### 5.1.6 Insufficient information

On occasion an estimation of the user story probably needs to leave “blank” i.e. un – estimated. It's better to ask for more information than to give an estimate that we are uncomfortable with. An estimate of 8 implies that it's a large story, but we expect it to take twice as long as a 4. Therefore, do not simply estimate ill – defined / bad stories as eights, because we will likely be expected to get it done in the same amount of time as it takes to get two stories estimated as fours completed. The goal of an estimation meeting is not to estimate all the stories; it's to provide estimates on the stories that provide sufficient information (Morris, 2006).

### 5.1.7 Estimation group size

Teams come in many different sizes. On smaller teams (6 or less) in general practice the entire team has to attend to the estimation session. The many points of view are likely to solidify vision and positively contribute to an estimate. If agile team is big, more than 8 members there is no need to participate for everybody in estimating each story. Additionally, it's an estimate, 6 people should be just as accurate as 15 people would be. If agile team is larger than 6 people, general practice suggests breaking into smaller groups for estimation. In general it should be enough to have at least 3 people to estimate any given story, but no more than 6.

### 5.1.8 No laptops

The main reason of this is concentration. At least no laptops for developers. Print the story list for everyone, or project the list on the screen, but do not ask the developers to read the story list from their laptops. Laptops almost always find ways to distract developers, thus taking away from the goal of the meeting: *Getting valuable estimates.*

### 5.1.9 Required participation

This suggestion is a very important one. In theory, no developer from outside the team should be attending an estimation session. That means that every developer that attends an estimation session will potentially be tasked with working on a story that's being estimated. If a developer is not comfortable estimating a story, then it means that (s) he not comfortable working on the story. Of course, there are exceptions. In general if team consist with new team members, it’s wise to give them one week to come up to speed before we could incorporate them into an estimation session. But, in general, a developer who refuses to participate in estimation should signal that there's a bigger issue that needs to be resolved (McDonald, 2009).

### 5.1.10 Stale estimations

Teams change, projects change, and random events occur. Whatever the reason, estimations can get stale. Stale estimations do not help anyone. The development team feels pressure to deliver to stale estimates and the business expects stories to be completed according to projected velocity. It does not matter why estimates get stale, what matters is that the estimates are no longer realistic and the plan is no longer reliable. It's better to admit that an estimation is stale than it is to plan with inaccurate information. For this reason, is wise to suggest revisiting any estimation that was given more than 12 weeks ago (Morris, 2006).

#### 5.1.11 Bribes

This is the easiest suggestion of all: Bring high quality snacks to all estimation meetings. Sugar has been scientifically linked to happiness, and happiness leads to collaboration. It's the simplest and cheapest possible way to make an estimation meeting something to look forward to. Keep in mind though, high quality is the key. If you bring the same snacks that are already sitting in the team room, it's not very exciting (Logue, 2008).

## 5.2 Planning poker

*The poker player learns that sometimes both science and common sense are wrong; that the bumblebee can fly; that, perhaps, one should never trust an expert; that there are more things in heaven and earth than are dreamt of by those with an academic bent. (David Mamet)*

Agile Planning Poker is generally attributed to James Grenning in 2002. It a general practice for the estimating user stories among agile teams. Planning poker combines expert opinion, analogy, and disaggregation into an enjoyable approach to estimating that results in quick but reliable estimates.

The process involves the members of the team independently developing quick Effort estimates and then comparing their estimate, discussing the differences and arriving at a consensus based on the discussions (Grenning, 2009).

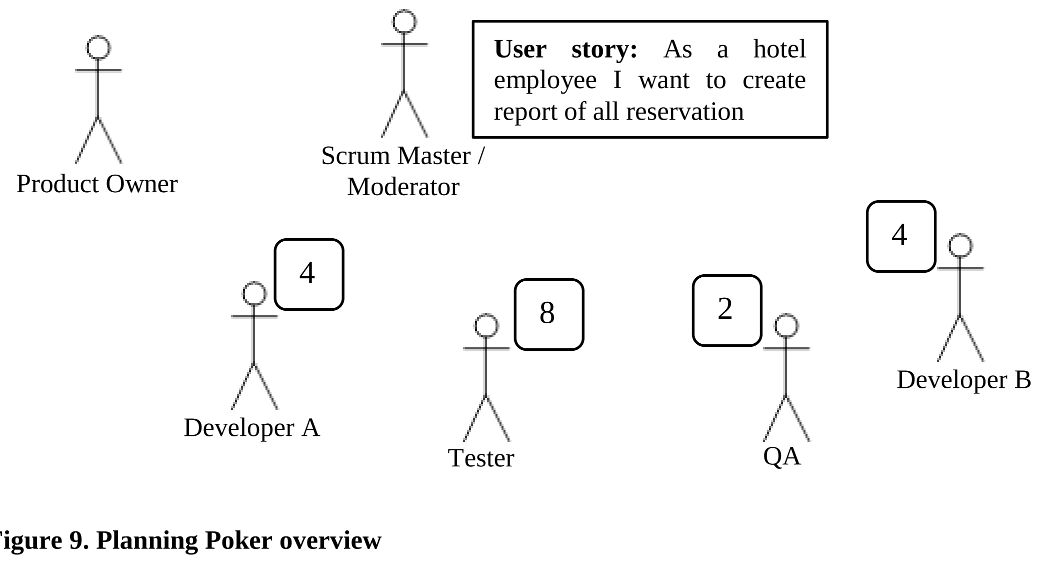

**User story:** As a hotel employee I want to create report of all reservation
Product Owner
Scrum Master / Moderator
4
Developer A
8
Tester
2
QA
4
Developer B

**Figure 9. Planning Poker overview**

Participants in planning poker include all of the developers on the team. Remember that developers refer to all programmers, testers, database engineers, analysts, user interaction designers, and so on. On an agile project, this will typically not exceed ten people. If it does, it is usually best to split into two teams. Each team can then estimate independently, which will keep the size down. The product owner participates in planning poker but does not estimate. At the start of planning poker, each estimator is given a deck of cards. Each card has written on it one of the valid estimates. Each estimator may, for example, be given a deck of cards that reads 1, 2, 4, 8, 16, 32, 64, and 128. The cards should be prepared prior to the planning poker meeting, and the numbers should be large enough to see across a table. The process relates to the game of poker only in the sense that the participants make their assessments discreetly, concealing them from the other participants until they all show their hands together. It is Important to the process that the participants conceal their assessments from each other and all participants reveal together. While this may seem silly, it serves the definite purpose of having each person arrive at their own conclusion without being influenced by the thinking of other members. The product owner answers any questions that the estimators have. The goal in planning poker is not to derive an estimate that will withstand all future scrutiny. Rather, the goal is to be somewhere well on the left of the effort line, where a valuable estimate can be arrived at cheaply (Lant, 2010).

After all questions are answered, each estimator privately selects a card representing his or her estimate. Cards are not shown until each estimator has made a selection. It is very likely at this point that the estimates will differ significantly. This is actually good news. If estimates differ, the high and low estimators explain their estimates. As an example, the high estimator may say, “Well, to test this story, we need to create a mock database object. That might take us a day.” The low estimator may respond, “I was thinking we’d store that information in an XML file – that would be easier than a database for us.” The group can discuss the story and their estimates for a few more minutes. The moderator can take any notes (s) he thinks will be helpful when this story is being programmed and tested. After the discussion, each estimator re – estimates by selecting a card. Cards are once again kept private until everyone has estimated, at which point they are turned over at the same time. In general cases, the estimates will already converge by the second round. But if they have not, continue to repeat the process (Lant, 2010). The goal is for the estimators to converge on a single estimate that can be used for the story. It rarely takes more than three rounds, but continue the process as long as estimates are moving closer together. It is not necessary that everyone in the team turns over a card with exactly the same estimate written down. Main point is not absolute precision but reasonableness.

### 5.2.1 The Moderator

The role of the moderator in this process is essential. The moderator sets the tone and the pace for the discussion. The moderator is typically the Project Manager or Scrum Master. The moderator (Logue, 2008):

- Reads the stories;
- Maintains pace to make sure that the process does not bog down;
- Moderates discussion, tempers the naturally competitive nature of some to ensure that nobody dominates with their perspective and that people who may not normally voice their opinions are also heard;
- Maintains a focus on good results – not on individuals having the winning hand;
- Ensures that people do not cluster their estimates. Some team members may want to be seen as correct or at least not too different from the pack and always cluster their results in the middle i.e. always guessing 4/4. If your results follow a standard Gaussian distribution, your safest bet would always be 4/4 if your motivation is to

be with the pack, or to be viewed as being correct. Remember that although you are hoping to have a Gaussian distribution, it is neither your motivation nor that of the team. What you are really doing with this exercise is trying to discover the outliers – the Stories that are actually much larger or much smaller than originally thought when the Story was first written. Following the herd will not uncover the outliers and consequently you will be exposing your project and team to risk. The Moderator must watch for this behavior and encourage different viewpoints from all team members to make sure the outliers are uncovered;

- Helps the team to avoid Group think. If everyone always agrees, then you can be almost certain that you are doing something wrong. Diversity of opinion is essential and it must be encouraged. If there is too much consensus, the moderator should play Devil’s Advocate to stimulate alternative thinking (In taking such position, the individual taking on the devil's advocate role seeks to engage others in an argumentative discussion process. The purpose of such process is typically to test the quality of the original argument and identify weaknesses in its structure and to use such information to either improve or abandon the original, opposing position).

### 5.2.2 Participants

In general practice participants in planning poker include all of the developers on the team. Remember that developers refer to all programmers, testers, database engineers, analysts, user interaction designers, and so on. (Logue, 2008):

- Smaller groups are more likely to have a group biases that can become Groupthink. More people mean more opinions;
- Junior or newer team members learn a lot from these sessions. Experienced developers have insight that comes from years of experience and years of making and then recovering from mistakes. This experience is of incredible value to the team as a whole and can help to quickly elevate the capabilities of the entire team;
- Ensures that estimation is not inadvertently biased by the viewpoint, knowledge or experience of any one team member;
- When Product Managers / Owners are involved and participate in the estimation, they gain significant insight into, and a greater appreciation for the issues related to the development of each Story. Similarly, developers gain better insight into the views of the Product Owners. This can be very useful for creating and enhancing good team dynamics and communication.

### 5.2.3 The Five Steps

1. Moderator Reads Story Description

The moderator reads the story description to the team.

2. Ask Moderator Questions

The team asks the moderator questions about the Story and the moderator provides clarification about the Story. The moderator makes sure that the questions are relevant to the estimation of Complexity and Size. Q and A should be no longer than two or three minutes. Ideally you want to stay under two minutes per Story.

3. Estimate Cards (Fingers)

The moderator finalizes discussion on the Story and asks each team member to make their estimates of Complexity and Size (see previous blog post for details) and write down their estimates for both Complexity and Size on a piece of paper or on their notepads. Estimates are not to be revealed until everyone has made their individual determinations.

4. Discuss Low / High Estimates

If everyone has exactly the same results (highly unlikely) then there is not much to discuss other than the possibility that there is a groupthink happening or that something is being missed. Assuming that there are differences of opinion, discuss the high and low values for both Complexity and Size. The objective is to tease out the issues that are perhaps not being considered by all team members. This is where team diversity is essential. The broader the range of experience of the team members, the more value there is to the exercise (Morris, 2006).

5. Consensus or Pessimist Wins

Time box the discussion and move to a conclusion. You are attempting to gain consensus, but consensus is not always possible. Decide as a guideline how you will deal with situations where you cannot achieve consensus. You really have three options (Griffiths, 2007):

- Take the average. Probably it's not the best approach because it tends to bias the results towards the optimistic view and may lead to underestimation.
- Accept the most pessimistic estimate. This is my preferences as in my experience, estimation tends to be optimistic and to counter that bias, and the pessimistic view seems to balance things out. This is not a license for the people with a consistent "glass half empty" view to rule the day, but it is a means of counteracting people's generally, and naturally optimistic estimation tendencies. Your team may have a different bias so adjust accordingly.
- Defer discussion to a separate meeting on particularly gnarly ones. This is done as a last resort. It's best to keep the estimation in planning session. If you find a great deal of disagreement on many of the issues, then decide to consistently go with the pessimistic view and then review the estimates in your retrospective. Use that information to recalibrate.

### 5.2.4 When to Play Planning Poker

Teams will need to play planning poker at two different times (Cohn, 2005).

1. There will usually be an effort to estimate a large number of items before the project officially begins or during its first iterations. Estimating an initial set of user stories may take a team two or three meeting of from one to three hours each. Naturally, this will depend on how many items there are to estimate, the size of the team, and the product owner's ability to clarify the requirements succinctly.
2. Teams will need to put forth some ongoing effort to estimate any new stories that are identified during iteration. One way to do this is to hold a very short estimation meeting near the end of each iteration. Normally, this is quite sufficient for estimating any work that came in during the iteration, and it allows new work to be considered in the prioritization of the coming iteration.

Alternatively, Kent Beck suggests hanging an envelope on the wall with all new stories placed in the envelope. As individuals have a few spare minutes, they will grab a story or two from the envelope and estimate them. Teams will establish a rule for themselves, typically that all stories must be estimated by the end of the day or by the end of the iteration. It's good idea of hanging an envelope on the wall to contain un – estimated stories (Beck et al., 2001).

### 5.2.5 Why Planning Poker Works

*"Planning Poker is a good way to come to a consensus without spending too much time on any one topic. It allows, or forces, people to voice their opinions, thoughts and concerns."*
— Lori Schubring

Based on previous description and overview of the planning poker workflow there are some reasons why it works so well.

- Planning poker brings together multiple expert opinions to do the estimating. Because these experts form a cross – functional team from all disciplines on a software project, they are better suited to the estimation task than anyone else;
- A lively dialogue ensues during planning poker, and estimators are called upon by their peers to justify their estimates. This has been found to improve the accuracy of the estimate, especially on items with large amounts of uncertainty. This is important on an agile project because the user stories being estimated are often intentionally vague;
- Studies have shown that averaging individual estimates leads to better results (Hoest and Wohlin, 1998) as do group discussions of estimates (Jørgensen and Moløkken, 2002). Group discussion is the basis of planning poker, and those discussions lead to an averaging of sorts of the individual estimates;
- Finally, planning poker works because it's fun.

## 5.3 Summary

User story estimation is done in story points that are basic unit of the estimation. Story Points are not a measurement of duration, but rather a measurement of size / complexity. Advantages of the story point:

- Pure measure of size and complexity;
- Relative; longer shelf life;
- Independent of the estimator;
- Typically faster;
- Easier to work with;
- Studies show we are better at relative estimating.

Agile Planning Poker is a quick, simple way to estimate the Effort required completing your Agile Stories. It engages the team and encourages them to bring into the open issues that might impact Effort estimates that may not otherwise be discovered or considered. The process is quick, simple, and based on general experience, very effective.

Main steps in the planning poker workflow:

- Moderator Reads Story Description
- Ask Moderator Questions
- Estimate Cards (Fingers)
- Discuss Low / High Estimates
- Consensus or Pessimist Wins
  - Take the average.
  - Accept the most pessimistic estimate.
  - Defer discussion to a separate meeting on particularly gnarly ones.

Advantages of the planning poker technique:

- Planning poker brings together multiple expert opinions to do the estimating.
- A lively dialogue ensues during planning poker, and estimators are called upon by their peers to justify their estimates.
- Averaging individual estimates leads to better results.
- It’s fun.

# 6 Agile Estimation

*"Prediction is very difficult, especially about the future."*–Niels Bohr, Danish physicist

What is estimation? Estimation is the calculated approximation of the result which is usable even if input data may be incomplete or uncertain. Agile estimation breaks this logic away and always re – estimates a project after each iteration. It based of different value system, deviations are not deviations, and separate iterations are more accurate estimations. Agile estimation techniques will not remove uncertainty from early estimates, but they will improve accuracy as the project proceeds. This is true because agile estimation methods take actual work into account as the project progresses. Project's work mix may be diverse, but if you measure at an aggregate level you can still identify an average that you can use for estimating your capacity (Kang, 2010).

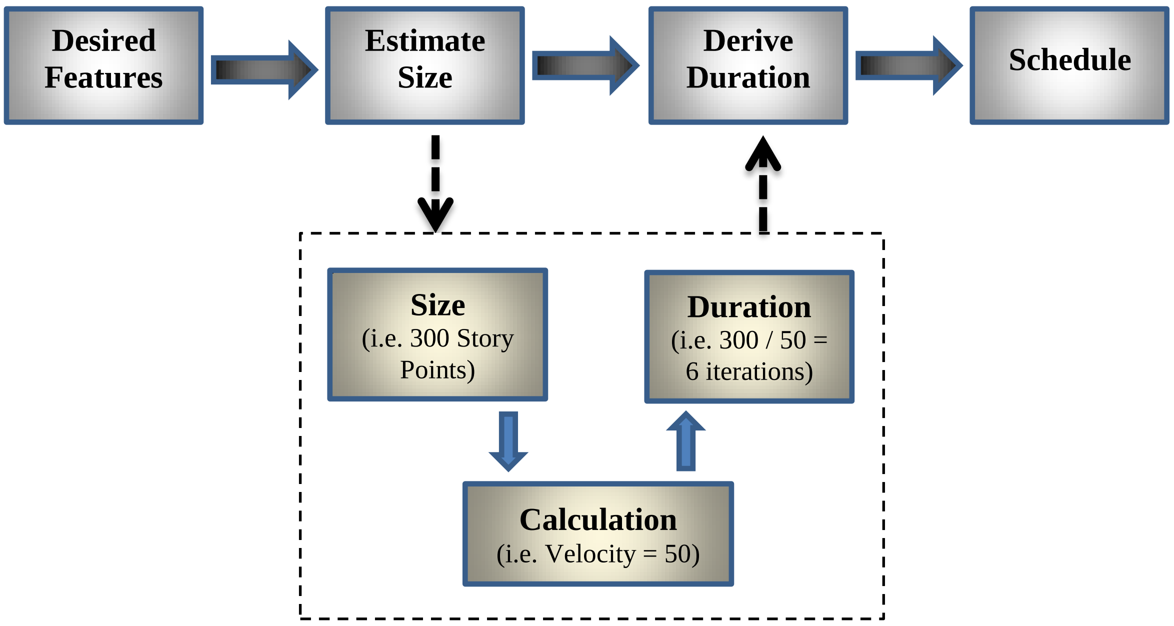


**Figure 10. Agile estimation overview**

- Desired Features – collect user requirements and convert them into user stories;
- Estimate Size – estimate size of the user stories, i.e. planning poker;
- Derive Duration – calculate duration of the implementation of the user stories based on the team velocity into iterations;
- Schedule – schedule entire project implementation.

Using story points to estimate how long a project will take works really great. One of the issues with using story points is that it's very tempting to directly relate them to hours — like saying a point is worth four hours. Such mapping between story points and hours has to be reviewed after each iteration it should not be the same constant through all iterations. Because after each iteration a teams' estimate becomes more accurate based on the experience from previous iterations. The team also estimates their capacity by resource type. Then the estimates, capacity, and known dependencies are entered into a project plan.

At this point, the team has a schedule that they feel confident in, and they share it with the stakeholders.

Agile estimation techniques address the shortcomings of this method. Based on Agile Manifesto you do not design and estimate all your features until there has been a level of prioritization and you're sure the features are needed. You used a phased approach to estimation, recognizing that you can be more certain as the project progresses and you learn more about the features.

At a high level, the phased process looks like this:

1. Estimate the features in a short, time-boxed exercise during which you estimate feature size, not duration.
2. Use feature size to assign features to iterations and create a release plan.
3. Break down the features you assigned to the first iteration. Breaking down means identifying the specific tasks needed to build the features and estimating the hours required.
4. Re-estimate on a daily basis during iteration, estimating the time remaining on open tasks.

Agile estimation derives at 3 scales:

- Iteration Plan Estimation
  Entire team gets together at the beginning of iteration, estimate each item (user stories) through planning poker session, based on team velocity stack user stories into the iteration.
- Release Plan Estimation
  Similar to iteration planning but extend the technique to multiple iterations. Tasks are generally coarser. Because it requires workable code for the customer, it would be wise to allocate perhaps entire iteration for the bug fixing in order to release "bug free" software. Release plan uses two basic techniques:
  - Given a release date, continue filling content until the date is reached;
  - Given content, continue adding iterations until content is completed;
- Project Estimation
  Similar to release planning but extend the technique to multiple iterations. Project plan is the last and main unit in an agile planning.

In an Agile environment, you increase the accuracy of your feature estimates by estimating the features together as a team. Estimates are not limited to managers or leads but also include developers, testers, analysts, DBAs, and architects. The features are viewed from various perspectives, and you merge these perspectives to create a common, agreed-on estimate (Cohn, 2005).

Entire – team estimation has additional benefits beyond diverse opinion. First, you get estimates from people who are closer to the work. Team members' opinions may be diverse, but they provide better estimates because they know existing code, architecture, and domains and what it takes to deliver in an environment.

A second benefit is team ownership of the estimate. If a manager provides the estimate, they hope the team supports the estimate and buys into it. If the team provides the estimate, they're immediately closer to owning the estimate, and they feel more responsible for making the dates they provided (Sehlhorst, 2011).

The three most common techniques for estimating are

- Expert opinion
- Analogy
- Disaggregation

Each of these techniques may be used on its own, but the techniques should be combined for best results.

## 6.1 Agile estimation workflow

Estimation agile workflow has 6 general units.

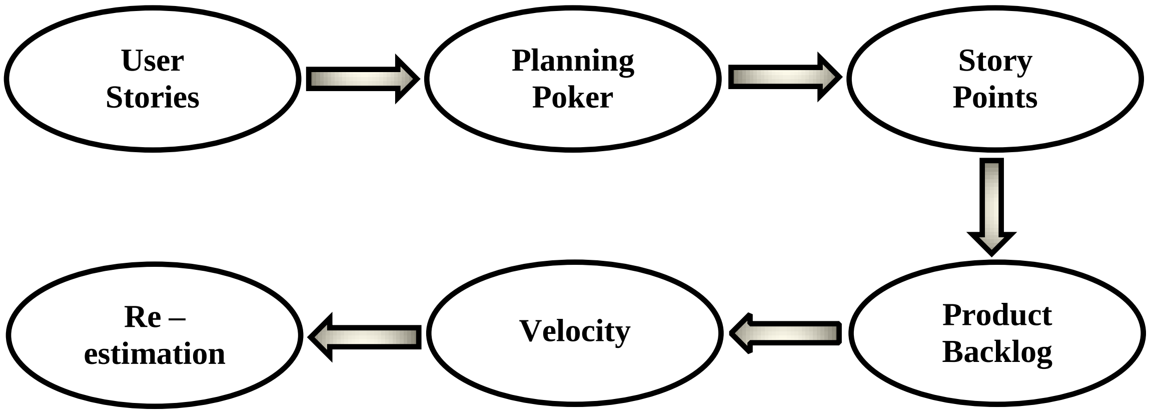


**Figure 11. Agile estimating workflow**

User breaks down the functionality into User Stories.

After all user stories are written, get a list of stories and do a high level estimate. Keep in mind it's **not** time based estimation.

Break down user stories to units of relative size in order to compare features. Start with 1 standard feature and then others features are 1x, 2x, etc. larger or smaller than that relative feature in size / complexity.

Product Backlog represents all of the tasks for the project (work items). Backlog has an item and its estimate; keep in mind that estimates is not time based, but point based. Backlog can also contain the priority (Stevens, 2008).

**Table 11. Product backlog**

| *Backlog item* | *Estimation* |
|---|---|
| As a user I want to make reservation | 1 |
| As a user I want to change conditions of my reservation i.e. dates, etc. | 2 |
| As a hotel employee I want to create report of all reservation | 4 |
| As a hotel administrator I want to add new employees to the system. | 4 |
| As a user I want to pay with credit card for my reservation | 8 |

Velocity is the number of story points per iteration completed. Calculate velocity to predict how much work to commit to in iteration. Keep in mind that velocity works only if you estimate story point's consistency. First iteration is usually as a guideline for the next iterations.

Re – estimation of the entire project has to be done after each iteration. Usually it's affected by:

- New velocity
- New story points added or removed

## 6.2 Successful estimation

Successful estimation in agile projects depends on various factors and conditions. Based on Mike Cohn book "Agile Estimating and Planning" some recommendations derive from:

- **Involve the whole team.** Primary responsibility for certain activities may fall to one person or group, as prioritizing requirements is primarily the responsibility of the product owner. However, the whole team needs to be involved and committed to the pursuit of the highest – value project possible. Based on many studies in agile estimation estimating is best done by the whole team, even though it may be apparent that only one or two specific team members will work on the story or task being estimated. The more responsibilities are shared by the team, the more success the team will have to share.
- **Plan at different levels.** It's quite important not to skip iteration plan while doing release plan. The release and iteration plans each cover a different time horizon with a different level of precision, and each serves a unique purpose.
- **Keep estimates of size and duration separate by using different units.** The best way to maintain a clear distinction between an estimate of size and one of duration is to use separate units that cannot be confused. Estimating size in story points and translating size into duration using velocity is an excellent way of doing this.

- **Re – estimate after each iteration.** Take advantage of the start of each new iteration to assess the relevancy of the current release plan. Remember, that after each iteration team’s velocity could change and new story point could be added or removed.
- **Track and communicate progress.** Many of a project’s stakeholders will have a very strong interest in the progress of the project. Keep them informed by regularly publishing simple, very understandable indicators of the team’s progress. Burndown charts and other at – a – glance indicators of project progress are best.
- **Acknowledge the importance of learning.** Because a project is as much about generating new knowledge as it is about adding new capabilities to a product, plans must be updated to include this new knowledge. As learn more about customers’ needs, new features are added to the project. As learn more about the technologies are using adjust expectations about rate of progress and desired approach.
- **Plan features of the right size.** Functionality that will be added in the near future (within the next iterations) should be decomposed into relatively small user stories – typically, items that will take one or two days up to no more than ten days. It will also provide stories that are small enough to be completed during one iteration for most teams. Otherwise use epic or themes for large user stories.
- **Prioritize features.** Work on features in the order that optimizes the total value of the project. In addition to the value and cost of features when prioritizing, consider the learning that will occur and the risk that will be reduced by developing a feature. Early elimination of a significant risk can often justify developing a feature early. Similarly, if developing a particular feature early will allow the team to gain significant knowledge about the product or their effort to develop it, they should consider developing that feature early.
- **Base estimates and plans on facts.** Whenever possible, ground the estimates and plans in reality. However, whenever possible, estimates and plans should be based on real, observed values. This goes, too, for an estimate of how much of a feature is complete. It’s easy to tell when a feature is 0% done (it have not been started), and it’s relatively easy to tell when the feature is 100% done (all tests pass for all of the product owner’s conditions / requirements of satisfaction). It’s hard to measure anywhere in between—is this task 50% done or 60% done? Because that question is so hard, stick with what it’s known: 0% and 100%.
- **Leave some slack.** Especially when planning / estimating iteration, do not plan on using 100% of every team member’s time. Just as a highway experiences gridlock when filled to 100% capacity, so will a development team slow down when every person’s time is planned to full capacity.
- **Coordinate teams through look ahead planning.** On a project involving multiple teams, coordinate their work through rolling look ahead planning. By looking ahead and allocating specific features to specific upcoming iterations, interterm dependencies can be planned and accommodated.

## 6.3 Expert Opinion

If it's needed to know how long something is likely going to take, ask an expert. At least, that's one approach. In an expert opinion – based approach to estimating, an expert is asked how long something will take or how big it will be.

Expert opinion derives these aspects:

- Rely on gut feel based on (extensive) experience;
- Disadvantage for agile: need to consider all aspects of developing the user story, so one expert will likely not be enough;

The expert relies on his / her intuition or gut feel and provides an estimate. This approach is less useful on agile projects than on traditional projects. On an agile project, estimates are assigned to user stories or other user – valued functionality. Developing this functionality is likely to require a variety of skills normally performed by more than one person. This makes it difficult to find suitable experts who can assess the effort across all disciplines. On a traditional project for which estimates are associated with tasks, this is not as significant of a problem, because each task is likely performed by one person. A nice benefit of estimating by expert opinion is that it usually does not take very long. Typically, a developer reads a user story, perhaps asks a clarifying question or two, and then provides an estimate based on his / her intuition. There is even evidence that says this type of estimating is more accurate than other, more analytical approaches (Kang, 2010).

## 6.4 Analogy

An alternative to expert opinion comes in the form of estimating by analogy, which is what we're doing when we say, "This story is a little bigger than that story." When estimating by analogy, the estimator compares the story being estimated with one or more other stories. If the story is twice the size, it is given an estimate twice as large. There is evidence that we are better at estimating relative size than we are at estimating absolute size (Griffiths, 2007).

Analogy derives these aspects:

- Relative to (several) other user stories;
- Triangulation: little bigger than that "3" and a little smaller than that "8"

When estimating this way, you do not compare all stories against a single baseline or universal reference. Instead, you want to estimate each new story against an assortment of those that have already been estimated. This is referred to as triangulation. To triangulate, compare the story being estimated against a couple of other stories. To decide if a story should be estimated at five story points, see if it seems a little bigger than a story you estimated at three and a little smaller than a story you estimated at eight.

## 6.5 Disaggregation

Disaggregation refers to splitting a story or feature into smaller, easier – to – estimate pieces. If most of the user stories to be included in a project are in the range of two to five days to develop, it will be very difficult to estimate a single story that may be 100 days. Not only are large things notoriously more difficult to estimate, but also in this case there will be very few similar stories to compare. Asking “Is this story fifty times as hard as that story” is a very different question from “Is this story about one and a half times that one?” The solution to this, of course, is to break the large story or feature into multiple smaller items and estimate those. However, it‘s need to be careful not to go too far with this approach (Griffiths, 2007).

Disaggregation derives these aspects:

- Break up into smaller, easier-to-estimate pieces/tasks;
- Need to make sure you do not miss any tasks;
- Sanity check: does the sum of all the parts make sense?

Estimation approaches (agile or traditional) can be divided into Heuristic (expert judgment based) and Parametric (calculation based) approaches.

## 6.6 Summary

Agile estimation is based of different value system, deviations are not deviations, and separate iterations are more accurate estimations. General components of the estimation:

- Desired Features – collect user requirements and convert them into user stories;
- Estimate Size – estimate size of the user stories, i.e. planning poker;
- Derive Duration – calculate duration of the implementation of the user stories based on the team velocity into iterations;
- Schedule – schedule entire project implementation.

Successful agile estimation derives from:

- Involve the whole team
- Plan at different levels
- Keep estimates of size and duration separate by using different units
- Re – estimate after each iteration
- Track and communicate progress
- Acknowledge the importance of learning
- Plan features of the right size
- Prioritize features
- Base estimates and plans on facts
- Leave some slack
- Coordinate teams through look ahead planning

# 7 Conclusions

This chapter summarizes the findings and provides possible future work. The findings are reviewed and discussed in section 7.1, whereas possible future work is discussed in section 7.2.

The main concerns that induced this research and identified problems are:

- Prepare correct user stories using INVEST model;
- Prioritize user stories;
- Evaluate risk and value, be ready to accept changing requirements in any phase of an iteration;
- Make an estimation based on these circumstances;
- What estimation techniques to use for the estimating user stories?
- How it fits into iteration plan?

## 7.1 Summarizing

The aim of this thesis is to investigate how a set of data derived from user stories, can be turned into a project estimate for use by developers. The thesis investigates best-practice methods to evaluate the data, using risk – and value estimation to prioritize features and applying changeability rate, and planning poker sessions to make the estimated project timeframe as close to the actual project timeframe as possible.

Agile development is more a way of thinking more than a pre – defined process. Flexibility, low cost and fast response of agile methods make most development teams realize that agile thinking is not only a good idea, but in many cases necessary in the fast – paced competitive environment of software development today. Agile methods are one of the most popular software development methodologies in modern software development, and used by most leading software manufacturers

User stories describe the benefits of products delivered by the project in a simple format. A user story describes desired functionality from the customer (user) perspective. This model of the user story is most often written like this: "As a [end user role], I want [the desire] so that [the rationale]. In summary, user stories document requirements with particular attention to the end user's point of view. Well-written User Stories are cornerstones for Agile Development. They should be independent of each other; the details should be negotiated between the users and the developers; the stories should be of value to the users; they should be clear enough for developers to be able to estimate them; they should be small; and they should be testable through the use of pre – defined test cases.

Velocity is an Agile (Scrum) term. It is a metric to estimate the productivity of the project team based on their previous work such as previous iteration.

In general, user story estimation is done in story points that are basic units of the estimation. Story Points are not a measurement of duration, but rather a measurement of size and / or complexity. Story Points are the most accurate way for estimating a user story by comparison, because story points are relative values. Probably one of the best ways for agile teams to estimate is by playing planning poker. Planning poker combines expert opinion, analogy, and disaggregation into an enjoyable approach to estimating. This results in quick but reliable estimates. Planning poker is a good way to come to a consensus without spending too much time on any one topic. It allows, or forces, people to voice their opinions, thoughts and concerns.

Estimating software is a mystery for most teams. Teams can spend huge amounts of time breaking down features to create their estimates, but the actual time needed is usually a vastly different number. The issue lies in two areas: techniques and expectations. Agile estimation techniques will not remove uncertainty from your early estimates, but they will improve your accuracy as the project proceeds.

## 7.2 Future work

Currently, literature on release planning estimation seems to be sparse, and difficult to find. Combining iteration estimations for each release hardly seems to be enough. Regression testing should for example be counted as a necessary stage of the estimation. Where release planning is crucial for successful delivery and customer satisfaction, resource limitations and effective factors need to be considered in a software systems development cycle.

Future work could include a closer look into this estimation approach in practice. One viable way could be to craft the model in a web – based project management tool, which could be handy to use for agile practitioners.

# References

Ahmed Nagy, Mercy Njima and Lusine Mkrtchyan. A Bayesian Based Method for Agile Software Development Release Planning and Project Health Monitoring. 2010 International Conference on Intelligent Networking and Collaborative Systems, pp. 192-199, 2010.

Band Om. Why Scrum Works. Available on – line: http://www.scrumalliance.org/articles/325-why-scrum-works. Date of publication: January 18, 2011. Date retrieved: January 26, 2012.

Bunio Terry. User Story Points and User Story Hours. Available on – line: http://blog.protegra.com/2011/04/08/user-story-points-and-user-story-hours/. Date of publication: April 8, 2011. Date retrieved: March 14, 2012.

Charette, R.N. Why software fails. IEEE Spectrum, 42, 42–49, 2005.

Cockburn, Alistair. Writing Effective Use Cases. Addison-Wesley, 2001.

Cohn Mike: User Stories Applied: For Agile Software Development. Pearson Education 2004.

Cohn Mike: Agile Estimating and Planning. Prentice Hall 2005.

Cohn Mike. Available on – line: http://www.userstories.com/products. Date retrieved: January 28, 2012.

Coutinho Rodrigo. SCRUM vs. Kanban. Available on – line: http://blog.outsystems.com/aboutagility/2010/11/scrum-vs-kanban.html. Date of publication: November 17, 2010. Date retrieved: January 26, 2012.

Duarte Vasco. What is Velocity? (In agile software development). Available on – line: http://softwaredevelopmenttoday.blogspot.com/2008/10/what-is-velocity-in-agile-software.html. Date of publication: October 23, 2008. Date retrieved: February 4, 2012.

Ferreira C. and J. Cohen. Agile systems development and stakeholder satisfaction: a South African empirical study. SAICSIT '08: Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries. New York, NY, USA: ACM, pp. 48–55, 2008.

Galin Daniel. Software Quality Assurance: From theory to implementation. Pearson Education 2004.

Grenning James. Planning Poker Party (The Companion Games). Available on – line: http://www.renaissancesoftware.net/blog/archives/36. Date of publication: February 6, 2009. Date retrieved: March 18, 2012.

Griffiths Mike. Agile Estimating – Estimation Approaches. Available on – line: http://leadinganswers.typepad.com/leading_answers/2007/11/agile-estimatin.html. Date of publication: November 20, 2007. Date retrieved: April 3, 2012.

Hearty P., N. Fenton, D. Marquez, and M. Neil. Predicting project velocity in xp using a learning dynamic Bayesian network model. IEEE Trans. Softw. Eng., vol. 35, no. 1, pp. 124–137, 2009.

Jeffries Ron. Essential XP: Card, Confirmation, Conversation. Available on – line: http://agileinaflash.blogspot.com/2009/03/card-conversation-confirmation.html. Date retrieved: January 22, 2012.

Kan Stephen H. Metrics and Models in Software Quality Engineering. Second Edition. Addison-Wesley 2003.

Kang Sungjoo, Okjoo Choi, Jongmoon Baik. Model-based Dynamic Cost Estimation and Tracking Method for Agile Software Development. 9th IEEE/ACIS International Conference on Computer and Information Science, pp. 743-748, 2010.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas (2001). "Principles behind the Agile Manifesto". Agile Alliance. Retrieved 6 June 2010.

Lant Michael. Estimate Story Size by Playing Agile Planning Poker. Available on – line: http://michaellant.com/2010/07/13/agile-planning-poker/. Date of publication: July 13, 2010. Date retrieved: March 18, 2012.

Lant Michael. Calculating the Velocity of Your Agile Projects. Available on – line: http://michaellant.com/2010/07/23/calculating-the-velocity-of-your-agile-projects/. Date of publication: July 23, 2010. Date retrieved: February 4, 2012.

Larman C. Agile and Iterative Development: A Manager's Guide. Pearson Education, 2003.

Larman C. Agile and Iterative Development – A Manager's Guide. Addison-Wesley, 2004.

Laudon Kenneth C., Jane Price Laudon: Management Information Systems. Pearson, 2009.

Lawrence Shari Pfleeger, Joanne M. Atlee. Software Engineering: Theory and Practice. Third Edition. Pearson International Edition 2006.

Lei Shen, Bei Jun Shen. Research and Practice of Agile methods. Computer engineering, 30-37, 219, 2005.

Logue K., K. McDaid. Agile Release Planning: Dealing with Uncertainty in Development Time and Business Value. 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 437-442, 2008.

McDonald Kent. AGILE PROJECT LEADERSHIP. What Does an Agile Project Plan Look Like? Available on – online: http://www.projectconnections.com/articles/120506-mcdonald.html. Date retrieved: March 12, 2012.

Misra V. K. S. C. and U.Kumar. Identifying some important success factors in adopting agile software development practices. The Journal of Systems and Software, vol. 82, p. 1869-1890, 2009.

Mnkandla E. and B. Dwolatzky. Defining agile software quality assurance. Software Engineering Advances, International Conference on, vol. 0, p. 36, 2006.

Morris Rob: Agile Estimation and Planning. CDL Systems 2006.

Moser Raimund, Pekka Abrahamsson, Ilenia Fronza, Jelena Vlasenko. Predicting Development Effort from User Stories. 2011 International Symposium on Empirical Software Engineering and Measurement, pp. 400-403, 2011.

Murphy Craig. Adaptive Project Management Using Scrum. Available on – line: http://www.methodsandtools.com/archive/archive.php?id=18. Date retrieved: November 16, 2011.

Sang DaYong. Requirement Analysis of Agile Development Process. Programmer. 2, pp. 70-75, 2009.

Schwaber, K. & Beedle, M. Agile Software Development with Scrum, vol. 18. Prentice Hall. 2001.

Schwaber, K. & Sutherland, J. Scrum guide. Development, 19, 21, 2010.

Sehlhorst Scott. Agile Estimation, Prediction, and Commitment. Available on – line: http://tynerblain.com/blog/2011/08/09/agile-estimation/. Date of publication: August 9, 2011. Date retrieved: March 18, 2012.

Soldi Emiliano. Prioritizing User Stories. Available on – line: http://www.emilianosoldipmp.info/2011/07/prioritizing-user-stories/. Date of publication: July 27, 2011. Date retrieved: March 4, 2012.

Standish Group. The Chaos Report 2009. Tech. rep., Standish Group, 2009.

Stevens Peter. Prioritizing the Product Backlog. Available on – line: http://agilesoftwaredevelopment.com/blog/peterstev/prioritizing-product-backlog. Date of publication: October 8, 2008. Date retrieved: March 3, 2012.

Wake Bill. INVEST in Good Stories, and SMART Tasks. Available on – line: http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Date of publication: August 17, 2003. Date retrieved: January 18, 2012.

Wang Ying, DaYong Sang, LiHua Wu. Agile Software Development Methods and Practices. XIDIAN University press, pp. 160-180, 2010.

XiaoYi Wang. Reconstruction Secret of Agile Developing. Software World, 21:59, 2007.

# 8 Appendix – Case Study

## 8.1 User stories

**Table 12. User stories [case study]**

| ***No.*** | ***User*** | ***Goal*** | ***Reason*** |
|---|---|---|---|
| 1. | Ordinary user | Login into the e – order system. | To see products for order. |
| 2. | Ordinary user | Make selections by manufacture, type, color. | To see product list based on selection. |
| 3. | Ordinary user | Use search field by product code / number. | To see product list based on search criteria. |
| 4. | Ordinary user | Get list of the products based on selection / search. | Product list consists of one editable field for the making orders. And price field is based on price percentage for the customer. |
| 5. | Ordinary user | Make an order of the product (s). | System subtracts ordered amount of the ordered products from the product‘s balance. |
| 6. | Ordinary user | Send an email with order details. | Email is send to ordinary user and administrative user. |
| 7. | Ordinary user | Keep order history | To see / check previous orders. Preview order details. |
| 8. | Ordinary user | Log out from the e – order system. | Close session. |
| 9. | Administrative user | Login into the e – order system. | To see products for order. |
| 10. | Administrative user | Make selections by manufacture, type, color. | To see product list based on selection. |
| 11. | Administrative user | Use search field by product code / number. | To see product list based on search criteria. |

| | | | |
|---|---|---|---|
| 12. | Administrative user | Get list of the products based on selection / search. | Product list consist of two editable columns. For the changing price and for changing product balance. |
| 13. | Administrative user | Customize price percentage for the customers. | To change price percentage for the customers. |
| 14. | Administrative user | Log out from the e – order system. | Close session. |

## 8.2 INVEST model evaluation

**Table 13. Evaluation of INVEST model [case study]**

| *User Story* | *Independent* | *Negotiable* | *Valuable* | *Estimable* | *Small* | *Testable* | *Total* |
|---|---|---|---|---|---|---|---|
| ID 1 | 5 | 2 | 2 | 5 | 5 | 4 | **23** |
| ID 2 | 2 | 4 | 5 | 4 | 5 | 5 | **25** |
| ID 3 | 2 | 2 | 5 | 4 | 5 | 4 | **22** |
| ID 4 | 3 | 2 | 4 | 3 | 4 | 4 | **20** |
| ID 5 | 2 | 2 | 3 | 4 | 4 | 4 | **19** |
| ID 6 | 2 | 4 | 4 | 5 | 5 | 5 | **25** |
| ID 7 | 3 | 3 | 5 | 4 | 4 | 5 | **24** |
| ID 8 | 5 | 1 | 1 | 5 | 5 | 5 | **22** |
| ID 9 | 5 | 2 | 2 | 5 | 5 | 4 | **23** |
| ID 10 | 2 | 4 | 5 | 4 | 5 | 5 | **25** |
| ID 11 | 2 | 2 | 5 | 4 | 5 | 4 | **22** |
| ID 12 | 2 | 4 | 5 | 4 | 4 | 5 | **24** |
| ID 13 | 1 | 4 | 5 | 4 | 3 | 4 | **21** |
| ID 14 | 5 | 1 | 1 | 5 | 5 | 5 | **22** |

## 8.3 Risk and value evaluation

**Table 14. Risk and value evaluation [case study]**

| *User Story ID* | *Risk* | *Value* |
|---|---|---|
| User story 1 | 3 | 4 |
| User story 2 | 2 | 9 |
| User story 3 | 2 | 9 |
| User story 4 | 3 | 7 |
| User story 5 | 5 | 6 |
| User story 6 | 5 | 7 |
| User story 7 | 6 | 9 |
| User story 8 | 3 | 2 |
| User story 9 | 3 | 4 |
| User story 10 | 3 | 9 |
| User story 11 | 3 | 9 |
| User story 12 | 5 | 9 |
| User story 13 | 4 | 10 |
| User story 14 | 3 | 2 |

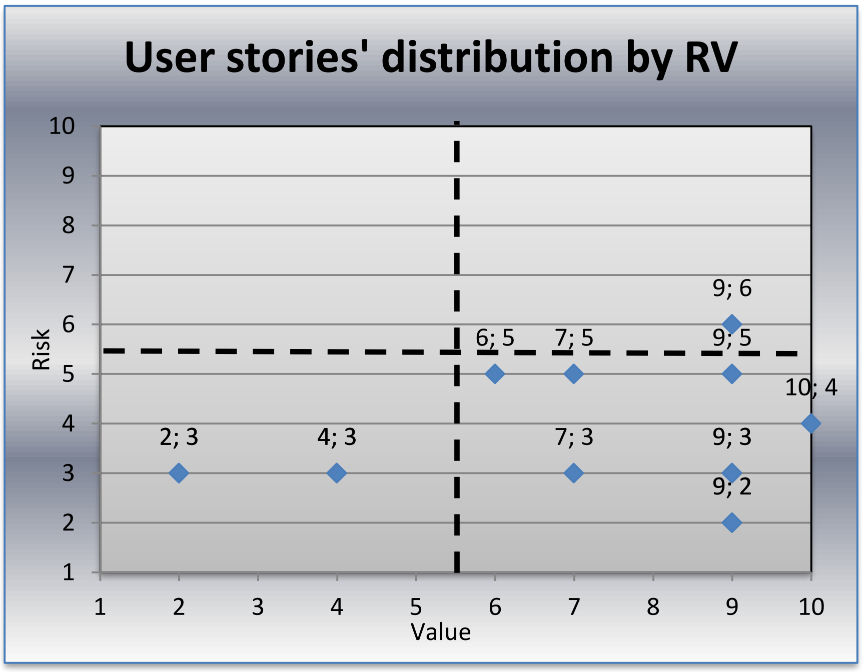


**Figure 12. User stories' distribution by RV [case study]**

## 8.4 Changeability rate

$$ChR = \left(\frac{R}{R_{Max}} + \frac{V}{V_{Max}} + \left(\frac{INVEST_{Max} - INVEST}{INVEST_{Max}}\right)\right) \times 0.25 \qquad (1)$$

$$ChR' = \left(\left(\frac{R}{R_{Max}} + \frac{V}{V_{Max}}\right) \div 2\right) \times 0.25 + \left(\frac{INVEST_{Max} - INVEST}{INVEST_{Max}}\right) \qquad (2)$$

**Table 15. Changeability rate [case study]**

| *User Story ID* | *Risk* | *Value* | *INVEST* | *ChR* | *ChR'* |
|---|---|---|---|---|---|
| User story 1 | 3 | 4 | 23 | 0.23 | 0.32 |
| User story 2 | 2 | 9 | 25 | 0.32 | 0.3 |
| User story 3 | 2 | 9 | 22 | 0.34 | 0.4 |
| User story 4 | 3 | 7 | 20 | 0.33 | 0.46 |
| User story 5 | 5 | 6 | 19 | 0.37 | 0.5 |
| User story 6 | 5 | 7 | 25 | 0.34 | 0.32 |
| User story 7 | 6 | 9 | 24 | 0.43 | 0.39 |
| User story 8 | 3 | 2 | 22 | 0.19 | 0.33 |
| User story 9 | 3 | 4 | 23 | 0.23 | 0.32 |
| User story 10 | 3 | 9 | 25 | 0.34 | 0.32 |
| User story 11 | 3 | 9 | 22 | 0.37 | 0.42 |
| User story 12 | 5 | 9 | 24 | 0.4 | 0.38 |
| User story 13 | 4 | 10 | 21 | 0.43 | 0.48 |
| User story 14 | 3 | 2 | 22 | 0.19 | 0.33 |