Haaga-Helia
University of Applied Sciences

Applying Behaviour Driven Development to enhance agile software development processes.

Gábor Munkácsi

Master's Thesis
Degree Programme in Information Systems Management
2018

Haaga-Helia
University of Applied Sciences

Abstract

25/11/18

Author(s)
Gábor Munkácsi

Degree programme
Information Systems Management

Thesis title	**Number of pages and appendix pages**
Applying Behaviour Driven Development to enhance agile software development processes.	**32**

Behaviour Driven Development (BDD) is a powerful tool to align business with software development. It enables the companies to be leaned and more agile, and to enhance Test Driven Development at the organization.

BDD affects the whole development workflow. It helps to divide the customer feature request to small, well defined deliverables, and communicates the business value across the whole flow. It enhances the collaboration between the stakeholders to be able to provide high valued, quality products for customers.

Implementing BDD is a massive change for big organizations. This change needs to be carried out with powerful change management and leadership.

BDD is compatible with SAFe which is one of the most popular scaled agile frameworks for big enterprises. Feature grooming and story planning is a key activity is SAFe also. BDD is a good tool to take the maximum out from SAFe.

In this thesis I describe an agile software development workflow, guided by BDD for complex products. I show how BDD improve collaboration and deliver value for all stakeholders. This document serves as a good theoretical background and as a starting point to introduce BDD at bigger software development organizations.

Keywords
BDD, TDD, SAFe, Agile Software Development, Lean Software Development

Table of contents

1 Introduction

Quality is defined as the main differentiator in telecommunication industry. High **quality** is essential. Anomalies in reliability may lead to service outage and profit loss.

Telecommunication networks are very diverse, vast and rapidly developing area. Telecommunication Research and Development must be sure to work on features what is **valuable for customers**. Engineers has to work on the right thing in the right time.

Short development lead time is the main factor for companies to acquire competitive advantage over competitors. Profit needs to be realized as fast as possible and it needs to be re-invested to enhance development velocity.

The goals above can't be reached without **efficient communication**. Smooth, solid and reliable communication channels need to be in place.

The purpose of the project to define how to apply BDD in the current development workflow to
- improve quality,
- enhance delivery of value for customers,
- shorten development lead time,
- and improve internal communication.

1.1 Scope

Scope of the thesis would focus on the work of Specification, Development, and System Verification triple on operational level. I concentrate on the development of highly complex systems.

The thesis will focus on the spreading of information about the business value from the perspective of testing. It is because BDD is also a test-driven development approach, and it is closer to my area of profession.

In the thesis work, I won't describe deep details of agile programming. My interest is to provide the needed information to development by BDD. The concrete techniques, how to turn this information to working code, is not part of this thesis.

1.2 Motivation

Current organization has separated locations for different functions. Architecture, development and system verification located in different continents and divided by cultural differences. Communication usually is done on poor communication channels, like requirement lists. We recognised that poorly worded requirements are understandable only by the originator side, but the information perceived in a different way on the other side.

We need a comprehensive and structured way of communication of business value and customer expectations. We need to implement a process which ensures that the receivers also understand the importance and value behind the requirements.

1.3 Objectives

My objective is to enhance the current development workflow to a more agile way of working by implementing Behaviour Driven Development. I would like to show, how BDD can be fit into agile frameworks like Scrum or SAFe. I would like to show how BDD can improve the collaboration and break up silos in the organization as show on Figure 1.

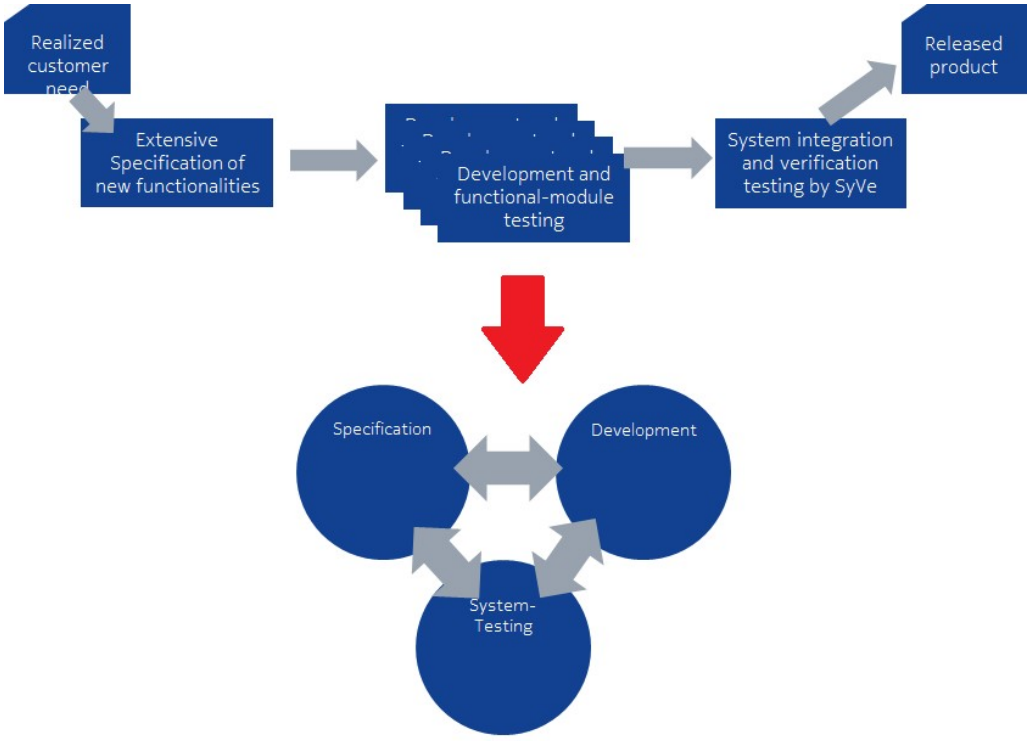

Figure 1: Transform silo based processes to agile workflow

I would like to show how BDD can lead the software development of complex systems. and would like to describe how BDD improve the following key factors:

- Shorter release cycles to be on the market as early as possible
- Eliminate waste by keeping the focus on the most valuable features
- Building up collaboration between the stakeholders from business to development
- Integrate test automation and documentation and releasing activities close to code development.

My aim is to give a detailed guide for implementing BDD, highlight the impediments and describe how BDD can be integrated and drive software development. This should serve as an input for the first pilot of introducing BDD at my company.

2 Theoretical framework

2.1 Behaviour Driven Development

Behaviour Driven Development (BDD) is a very popular phrase in software development nowadays. In many places BDD is defined the same as using Gherkin(Rose, Wynne, 2015) and define test scenarios in Given/When/Then form. BDD is more like a concept of connecting business with development and help the customer value to spread into the development activities.

Gherkin is an open specification format what gives a structured, human readable and executable form for scenarios in plain text. (Seb Rose, 2015)

In “The BDD books” by Gáspár Nagy and Seb Rose, 3 practices defined (Gáspár Nagy, 2018):

- Discovery
- Formulation
- and Automation

Discovery is a structured, collaborative, activity that uses concrete examples to establish a common understanding of the participants, uncover ambquities and misunderstandings. Proper discovery, with the participation of business representatives (most likely a Product Owner), developers and testers help to eliminate misunderstandings in early phases and do things right the first time.

Formulation helps to put the examples in a commonly understandable format and language. This can be varied by domain, and culture, but Gherkin gives a good starting point.

Automation of acceptance tests helps to shorten the feedback loops for developers, by the automated execution of tests.

Usually automation turns to be the less difficult part in software developer companies where the necessary coding knowledge is already present. The collaborative work and establishing a common language for business and development stakeholders is a much harder. Usually the necessary communication channels and practices are not present, and the customer value is lost during the many handovers over long process flows.

In my thesis project I would like to focus mainly on the discovery part. My aim is to put BDD in practice and enhance the collaborative work between the different function groups.

My company is currently in the middle of an agile transformation. BDD can be a valuable tool which helps the inter-functional communication and give guidelines for a collaborative way of working.

2.2 Acceptance testing

There are different ways to classify testing, test-levels. ISTQB defined 4 level (Graham, 2008):

- Component testing
- Integration testing
- System testing
- Acceptance testing

These test levels are based on the complexity of System Under Test (SUT) what they cover. Only in Acceptance testing appears the behaviour of the whole system, and the business perspective. However, behaviour approach should affect all the testing levels.

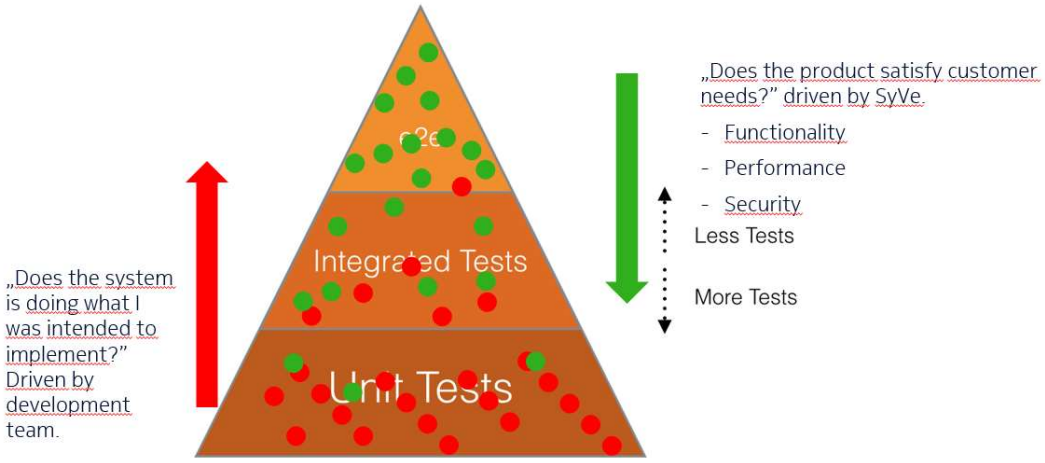

Figure 2: Acceptance tests Vs.developer tests

In my point of view: there are 2 different purpose of testing:

- The first one is to check if the system is doing what was the developer intention to do. These test cases have a bottom-to-up approach, usually contains a high number of Unit and module test cases, and fewer of System integrated or end-to-end. These test cases can be created by the developers themselves, they are simpler and easily can be automated. The actual form is highly dependent from the technical details and architecture. Let's call them Developer tests.

- The purpose of the other group of test cases is to verify if the business value is provided by the product, and the final solution will satisfy the customer needs. These have a top-to-down approach, derived from business requirement. Technical and architectural implementation is not that important. They are more complex, the execution time usually higher and sometimes it is not possible, or feasible to automate. They are responsible to cover Non-functional requirements (Performance, Security, etc…). I will call these kind of approach as System Verification Acceptance tests.

The importance of independent testing from coding is increased in the second type of test cases. If the customer value is misinterpreted by the developer, the independent tester needs to catch the issues.

BDD is a Test-Driven Development (TDD) method. BDD shows how to create the test cases from the business requirements. These test cases are usually end-to-end or system verification test cases. If we stick to formulating our tests from business (or the expected behaviour of the product), and we use them to drive, manage, and follow-up the development, we reach BDD through TDD.

2.3 Lean and Agile thinking

Lean thinking got its name from “The Machine That Changed the World: The Story of Lean Production” (Womack, 1990). This book describes the story of Toyota Lean thinking compared to Henry Ford’s standardized automobile parts and assembly techniques and management. Because, Software Development differs from production processes, M. Poppendieck (2003) translated lean principles and the 7 wastes to software development expressions in Principles of Lean Thinking.

Jeff Sutherland in “Scrum, The Art of Doing Twice the Work in Half the Time” (2014), gives a practical handbook for agile software development. He also mentions Lean thinking and describe principles of agile software development change management, which is aligned with Lean principles.

In Lean Software Development: An Agile Toolkit (2003), M. Poppendieck provided a set of 22 tools to implement Lean thinking in software development companies. This set of tools fit into the agile software development and management framework. It is a useful handbook, which helps to lean agile environment, make iterations shorter, and ensure delivery of customer value-added features.

In 1st tool she translates Lean principles and the 7 wastes of lean to software development:

Lean Principles in Software development:

- Add Nothing But Value (Eliminate Waste)
- Center On The People Who Add Value
- Flow Value From Demand (Delay Commitment)
- Optimize Across Organizations

7 Waste in software development:

- Overproduction = Extra Features
- Inventory = Requirements
- Extra Processing Steps = Extra Steps
- Motion = Finding Information
- Defects = Defects Not Caught by Tests
- Waiting = Waiting, Including Customers
- Transportation = Handoffs

In Making Technology Investments Profitable: ROI Roadmap from Business Case to Value Realization (2nd Edition), definitions of “Value leak” and “Being On-Value” are defined. These customer value is the driver of BDD.

The most common agile methodology is Scrum. In Scrum the cross-functional development teams (7-11 people), working in 1 to 4 weeks iterations. Planning for 1 iteration and review it at the end to be able to continuously improve their processes. Scrum master serves the team to avoid any impediments, develop and empower the team members. Product Owner is responsible for the prioritizing of the work to do of the team and aligning the team objectives with the company goals. (Sutherland 2014)

BDD has benefits which get the company closer to a leaned and agile organization, but it also has some lean-agile prerequisites what are important for a successful introduction of BDD.

To “Add nothing but value” we, developers need to understand of customer value, what comes from business. BDD realize that, this need to be directly delivered to R&D with the shortest communication chain. It helps to avoid misunderstandings, and by that, eliminating the waste of extra, non-valuable features. It bypasses the extra steps between business and development and eliminating defects and decreasing lead time.

However, implementation of BDD can be very painful if the organization is not leaned enough or not agile enough to drop comprehensive documents, and prefer light weight specifications based on human interactions, and cooperation.

3 A software development workflow with BDD

BDD affects the whole feature development workflow. Figure 3 shows a leaned flow, how the business value is added to a new feature. The process is driven by BDD, and quality is built in.

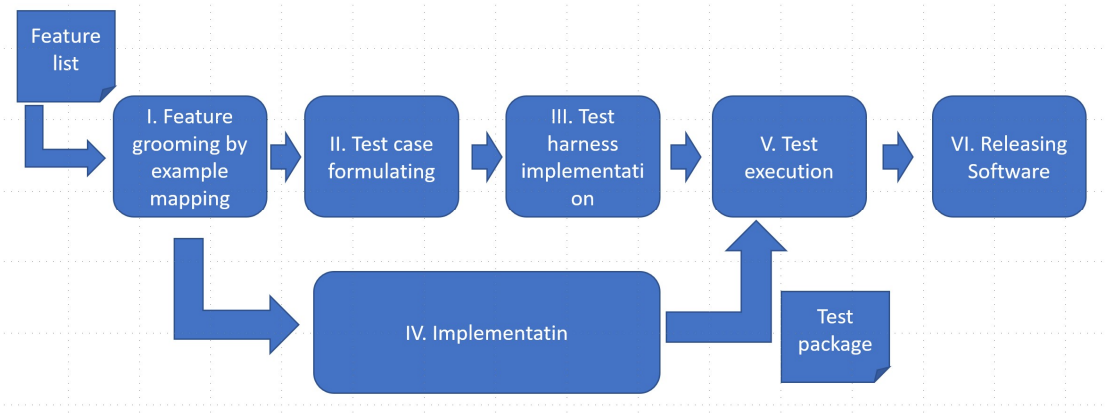

Figure 3: Proposed software development workflow

3.1 Feature List

The input of the workflow should be a prioritized in Feature backlog where elements ordered based on Weighted Shorted Job First (WSJF). (Reinertsen, 2009). WSJF counts with the Duration of a tasks and the Cost of Delay if the tasks are not done.

Figure 4 shows, how the Cost of Delay is decreased by ordering the delivery of features from short, high value tasks to longer, lower value tasks.

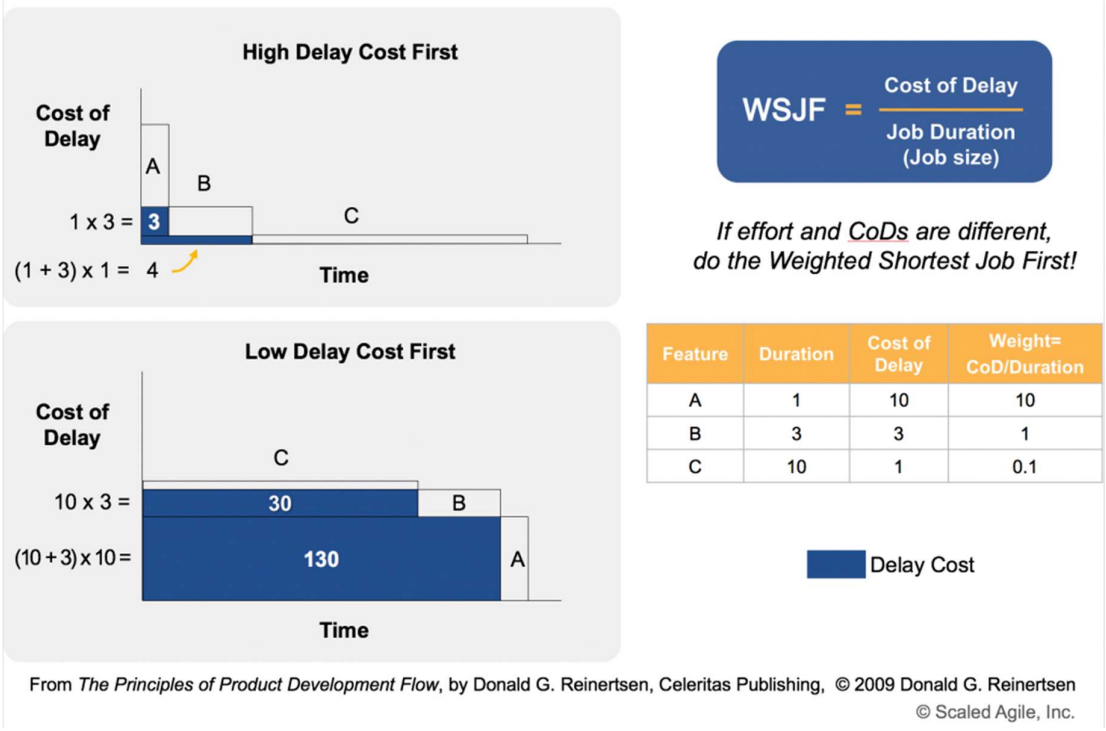

Figure 4: Applying the WSJF algoorithm delivers the best overall economics – 2009 Donald G. Reinertsen

Cost of Delay is the sum of Business value, the Time Criticality and Risk Reduction or Opportunity Enablement. Business value is the expected revenue impact on our business. It can be the profit what we realize, or a potential penalty what we can avoid by making the feature done. Time criticality is about how the business value decay over time. Is there a deadline, how far it is, is the customer tolerate delays? Risk reduction-opportunity enablement value are e.g.: the information what we receive of implementation, or the potential future businesses open.

In some cases, we might don't have enough understanding of the new feature to be able to estimate the duration properly. In this case we can use story points. Story points represents:

- Volume: How much work needs to be done?
- Complexity: How hard it looks based on current knowledge?
- Knowledge: How much do we know?
- Uncertainty: How much we don't know?

Backlogs usually contains several elements to choose from. We don't need to define these numbers accurate. It is enough if we can compare the different elements to each other.

The specification or planning of a feature is also a time-consuming activity. It is very important to save our effort already from the beginning. Practising BDD also takes efforts from the employees. Use this prioritized list as the input for the workflow. It is just a waste of time to execute example mapping or other BDD activities on low priority features, when there are still higher priority elements.

3.2 Feature grooming by example mapping

The purpose of the Feature Grooming meeting is to get better understanding of the feature and to divide it to smaller deliverables. A good way to reach these goals is the Example Mapping. (Rose, Wynne, Hellesoy, 2015)

Example mapping is the starting point of BDD. The input is the highest priority feature, the corresponding business rules and user stories and the knowledge and expertise of the business owner, product owner and the development team.

This meeting should be a collaborative, structured meeting between Product owner (Feature owner), developers and testers. The meeting should be short, hold frequently in small groups. Not all the group members need to be there, but at least one representative from each area.

Before the meeting, the product owner chooses the first element from the feature backlog. Divide it into user stories and select the first few what can fit in a 30-45 minutes meeting. PO sends out the scope of the meeting 1 day before, to give time for preparation.

The PO starts the meeting by selecting the first user story. Writes it on a yellow post-it and put it on the board. The team starts a conversation and bring up examples. The examples should be worded on business language and from user perspective. It usually describes the starting state, some user actions and the expected behaviour of the system.

During the conversation, the team may use different tools to help better understanding. For example: flow charts, UI snapshots, state diagrams or user personas describe different user behaviours.

After an example is defined, the facilitator writes it on a green post-it and put it under the user story. By having more and more examples, the team can define generic rules or requirements. The facilitator writes them on blue post-it and put above the examples. Rules helps to organize the examples into groups.

During the conversation there might be some questions come up, where it is not easy to find the best solution, and it would take more time to analyse it. If the team stucks, the facilitator stated the issue in a red post-it and put it into the corner. The team should not waste time from the meeting to find solution for every question. These issues can be addressed later.

At the end of the meeting the group reach the example map of the selected user stories. The participants have a common understanding of the expected behaviour and the problematic parts are also defined, where further investigation needed.

As mentioned in the previous chapter, it is not needed to groom all the features in the backlog. Keep a limited number of features groomed in the top of the backlog which brings enough development work during the following iterations. There is a high chance, that the elements in the bottom half of the backlog turn to be obsolete before the more important features can be finished. Or, new more important features turn up by the time.

Example mapping is described in detail in Discovery: Explore behaviour using examples by Gáspár Nagy and Seb Rose. (Nagy, Seb, 2018)

3.3 Implementation

After example mapping meetings all the stakeholders have the common understanding. Developers know what to deliver, and most of the questions are answered. They understand the business value what they need to provide.

As a part of implementation, developers also need to write and execute testcases. These tests are focuses on „Does my code do the same as my intention was to do?”. These tests are most probably unit tests or low level, module focused functional tests.

Developers working parallel as the System Verification team. Both team should focus on the same features, user stories. When the developer team(s) is confident about the quality of their code, and their own test cases are passed, they build the test package and provide it for test execution.

In best scenario, the automated System Integration or Acceptance test sets are available by that time. With a well-designed continuous integration and automated test execution system, developers should get fast feedback from the results of System Integration tests.

The responsibility of the implementation team has to be to provide testable software. Usually the interaction with the system is done on a Graphical User Interface (GUI), which is usually java or web based or on Application Programming Interface(API).

There are a few good automation frameworks to automate GUI interactions. For example: Selenium for WEB applications. But, the different elements on the GUI must be well identified. Otherwise it will be cumbersome to automate the tests. If the implementation team gives attention to provide well defined identifications for the element, the test automation work can be easy.

Implementation team also needs to provide good APIs for the system. For example, to reach the desired starting state to execute a test. APIs should give a programmable fast interface for the backend, databases, etc... If this is not available, then the only way is to reach the starting state by sequential steps on the frontend which usually tie consuming and contains stability risks.

3.4 Test case formulating

To have automated test cases two steps needs to be done. Test planning in a structured format what is understandable by a computer. And, the implementation of the test harness which translates the test steps to executable program code. In the System Verification team two type of member characteristics are needed: a Verification engineer who has a good domain knowledge, understands the system as a whole, and know the business value. He knows the needed user interaction with the system to perform the defined example. Verification engineer doesn't necessarily have programming skills. On the other hand, the Automation Engineer has a programmer mindset. He needs to have good coding and scripting skills. They don't need to be aware of the whole end-to-end system. It is the perfect situation if both characteristic have in the same person, but usually the two job requires different personality. That's why it is better to handle Test Case formulating (Test planning) and Test harness implementation in a different step.

Keyword driven testing (Rose, Wynne, Hellesoy, 2015) is a good way to connect examples with executable test cases. ROBOT is a good framework to get the benefits of keyword driven testing. It is also quite worldwide. It allows us to embed keywords into each other to get a hierarchically structured test case.

Phrase „Formulate test cases" was created by Gáspár Nagy (Nagy, 2018) and will be described in The BDD Books: Formulation. It describes how to express the test cases in Given/When/Then form.

However, this form is too high level for automated test execution. I would introduce multiple level for a well-defined test case:

The top level, "Business Level" is formulated in the Given/When/Then form. It uses clear business, behaviour-oriented language. Here we don't care about the way of performing the test, don't care about the technical details. The test case states the current state(Given), the action (When) and the Expected behaviour (Then).

```
Scenario: Register Bob as a user
         Given: Bob is not registered as a user.
         When: Bob registers on the Web UI.
         Then: Bob registered as a user
```

On the next level (let's call it: "Interaction level") we need to describe "how to actually do it?". This is a level what describes the test case the best for manual execution. For example, in Given part: what steps the user needs to do to get to the desired state.

```
Keyword: Bob is not registered as a user:
         Get User list from API.
         If User List contains Bob
                  Delete user Bob by API
```

In When part: what elemental steps the user needs to do to perform the action.

```
Keyword: Bob registers on Web UI:
         Open browser
         Go to URL:https://service.com/
         Click Button Register new user
         Fill in User Name with "Bob"
         Fill in Password "1234"
         Click Button Register
```

In Then part: How can the user check if the desired expected result is really happened.

```
Keyword: Bob registered as a user:
Get User list from API
If User List contains Bob
```

```
        Test Case Passed
Else
        Test Case Failed
```

3.5 Test Harness Implementation

The third level is optional. It is a technical level which contains the realization of technically hard keywords. For example, find an element in an ordered list, distinguish between more elements with the same id, or handle dynamic elements on a web UI.

The fourth level is the backend part. This contains the common libraries like Selenium. But usually, the available libraries are not enough for all the keywords, in this case other keywords needs to be implemented. For example, to open an SSH connection, execute bash scripts or open a RESTAPI session. The most common language is python.

The upper two levels are the playground of verification engineers, the bottom two is the test automation engineer's area, where the scripting skills become more dominant.

The test harness implementation enables the automated execution of the testcases.

3.6 Test Execution

Test execution must serve as a fast feedback for developers. Our test cases need to be divided for a fast executable, fast feedback set, and a slower but still necessary set from quality perspective. To be able to select the correct tests for execution, test cases need to be version controlled together with the code and executed on the relevant branch. Branches should contain the tests related to the developed feature and the regression set, what is common on all branches.

3.7 Releasing the software

When all the test cases (feature and regression) are passed on a feature branch, the feature is ready for releasing.

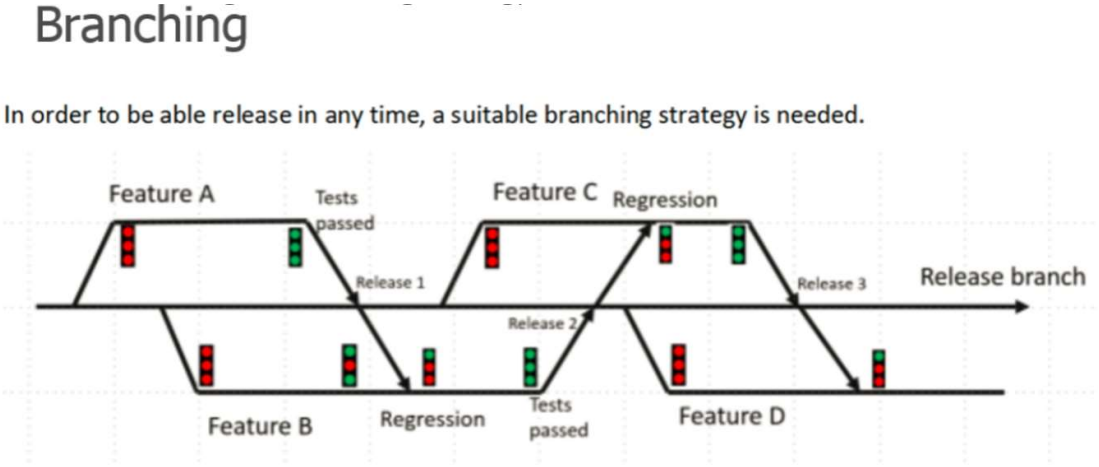

Figure 5: Testing in case of multiple branches

The branch must be merged back to the release branch. This action should be straightforward because the with the regression test set we validated that the feature branch still support the legacy features and the new on top of it. The branching strategy is illustrated on Figure 5.

To ensure the easy merge to the release branch and fast releasing of finished feature, we need to merge the new feature to the other branches. On “first wins” bases, this process is easy for the developers who finished their feature first, but it is an extra overhead for the other feature to merge and eliminate conflicts. It is important to work in small features and as less branches as possible, to minimize this overhead. Big features increase the complexity of merges. Number of branches multiplies the merge overhead.

3.8 Visualizing the work in progress on a Kanban board

Figure 6 shows a possible way to visualize how the agile stories or features flow through the different steps. Every features or ideas start in the funnel. There are no pre-requisites to list any new ideas here, only a title what describes the feature in brief. Every feature needs to be analysed. For example, describing the business case. Calculate the potential business value and prioritize compared to other features. After the prioritization the feature will be part of the backlog. If there is free resources, the grooming of the highest priority feature will be started, and the features will be splitted up to stories.

The stories will flow through parallelly on the testing and developing activities. During the coding, the automated acceptance tests are created also. If all the test cases are ready, and they are passed with the latest code, the stories are ready for releasing. In many

cases, releasing requires some extra tasks. For example, not-automated testing, documentation, legal requirements. All the stories, which are part of the same feature, should be ready also before they can be released.

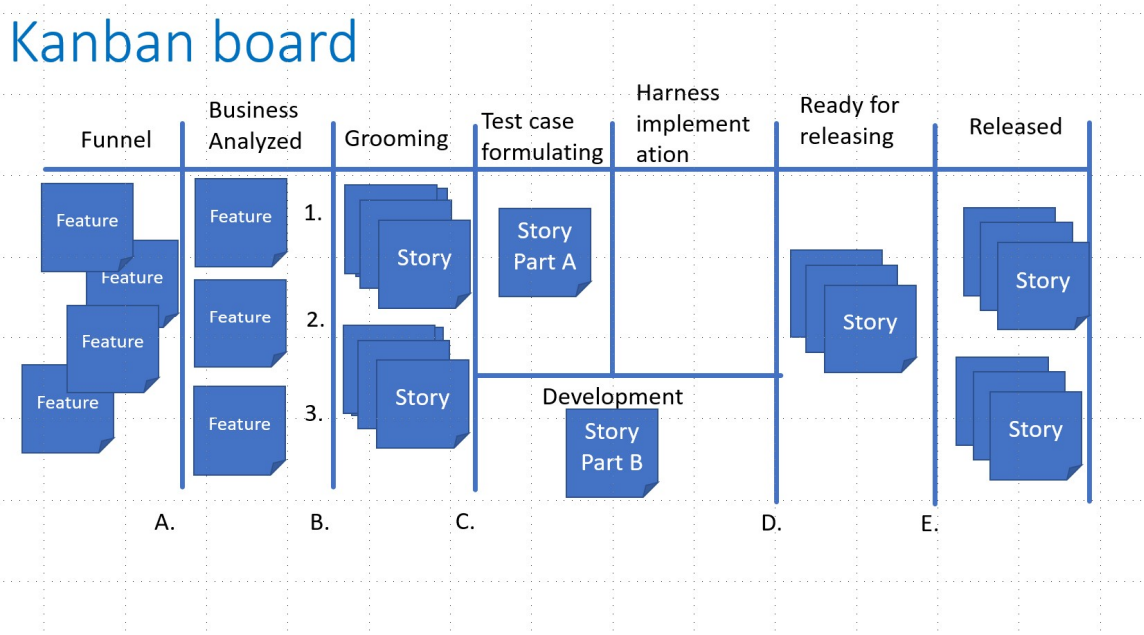

Figure 6: Visualize the work in progress on a kanban board

Between the steps, we should define clear Definition of Done criteria (DoD) what must be ready to move the feature or story to the next step.

For example:

- A:
  - Business case is defined.
  - Rough effort estimate defined
  - Business value clear
  - Priorities set
- B:
  - Feature assigned to a team.
- C:
  - All the stories are small enough for further work
  - Stories are clear and understandable for the development team
  - Team is committed to deliver the feature
  - Dependencies are defined and clear
- D:
  - Tests are ready, automated, and executable from CI
  - Code is ready, reviewed
  - Test cases are passed with the latest code.
- E:
  - All stories are ready in a feature
  - Features is documented
  - Product with the new code is satisfy other non-functional requirements.

The load of work can be managed by assigning “work in progress limits’ (WIP) for the different steps. These WIPs should be small enough to maintain good flow, and fast throughput. It can depend on the size of the team, and the level of competence in different areas.

4 BDD in SAFe®

At my company Scaled Agile Framework® (SAFe) is selected as the main agile framework to drive agile transformation. We are in the middle of the introduction, and the organization is committed to implement SAFe methodologies. So, it worth to examine, how to fit BDD in SAFe.

SAFe and Scaled Agile Framework are registered trademarks of Scaled Agile, Inc.

4.1 SAFe in a nutshell

"SAFe® is an online freely revealed knowledge base of proven, integrated patterns for implementing Lean-Agile development. It provides comprehensive guidance for work at the Portfolio, Large Solution, Program, and Team Levels."

Dean Leffingwell - Copyright © Scaled Agile, Inc.

SAFe is a collection of agile tools and methodologies integrated to a system to serve bigger companies. Not just focusing to the team level, but scale it up to program, solution and enterprise level. The full configuration of SAFe is shown on Figure 7.

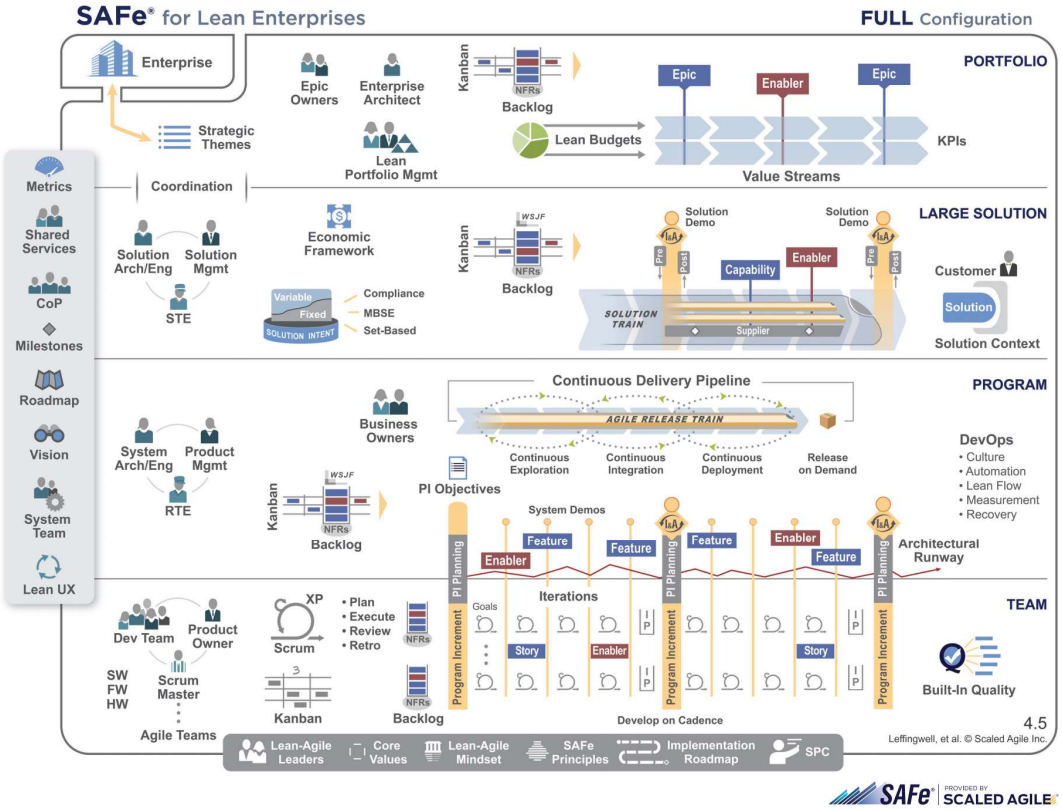

Figure 7: SAFe for Lean Enterprises (© Scaled Agile Inc, 2018)

On team level SAFe works in common scrum teams. Usually 2 weeks iterations starting with Iteration planning meeting, and finish with Iteration Demo and retrospective. The team objectives are present for every iteration, and progress is followed on daily scrum meetings. Scrum master facilitate the meetings and serve the team. Product owner keep the backlog up-to-date and align the team tasks with common goals. Team job is followed by stories.

On program level, agile teams organized in Agile Release Trains (ART). Teams are synchronised by Program Increments (PI): 4 to 6 iterations. Every PI starts with PI planning event and ends with Inspect and Adapt meetings. A Release Train Engineer keeps the system moving forward. Product Management keeps the program backlog up-to-date and prioritize features. System architects keeps the Architectural Runway low, and responsible for e.g.: the software architecture and main guidelines. Business Owners represents the customer and make the key stakeholders to understand the business context.

Systems teams provide the common artifacts for the ART. They are working on the Continuous Delivery Pipeline, automate measuring tools, automate workflow, visualize the work done in ART or responsible for System Verification.

Program and Team levels are essential for SAFe. Large Solution and Portfolio Layer is for bigger enterprises. These levels align the strategic decisions with agile development. It defines the high-level value stream, lean budgeting and vision of the enterprise. BDD mainly affects the levels from Teams to Large Solution.

4.2 Lean-Agile mindset in SAFe

SAFe culture is based on the House of Lean:
The foundation is the Lean-agile leadership, where 4 pillars stand on (see Figure 8):

- Respect for People and Culture
- Flow
- Innovation
- Relentless Improvement.

Figure 8: The SAFe House of Lean (© Scaled Agile Inc, 2018)

These pillars support to reach the goal: “The goal of Lean is to deliver the maximum customer value in the shortest sustainable lead time while providing the highest possible quality to Customers and society as a whole. High morale, safety, and customer delight are additional goals and benefits.” (https://www.scaledagileframework.com/, 2018)

SAFe reviewed the agile principles applied for software development by other authors, and a formed the SAFe agile principles for general development context:

1. Take an economic view
2. Apply systems thinking
3. Assume variability; preserve options
4. Build incrementally with fast, integrated learning cycles
5. Base milestones on objective evaluation of working systems
6. Visualize and limit WIP, reduce batch sizes, and manage queue lengths
7. Apply cadence, synchronize with cross-domain planning
8. Unlock the intrinsic motivation of knowledge workers
9. Decentralize decision-making

4.3 Agile stories vs examples

As described in chapter 3.2, features will be groomed to smaller features, stories, and smaller stories. These stories should be in the for:

"As a <user role)>, I want <activity> to, so that <business value>."

The story is in the form of "user form". Our goal is to articulate the valued functionality from the user perspective. The technical aspects are not part of the story.

However, to be able to understand the details, how the story should be delivered, we need to define examples. Examples give answers for the details in the functionality what is also very important and helps to take all stakeholders to the same understandings. A good form of the examples is the Given/When/Then form, mentioned in Chapter 3.4.

"Given <Context>
When <Action>
Then <Expected behaviour>"

Examples should be added to the stories to make them understandable and to define some technical details also. For example, we can define the story "Failed login" as this:

"As a user I want to authenticate myself by a username and password, so that my data is not accessible by others"

Examples can be added by multiple people: Program owner, Security specialist, Developers, tester, or even the customer itself. For example, the customer can define where should we get after successful login:

"Given, the login page shown,
When, user gives valid credentials,
Then, the greeting page is shown to the user"

Later, one of the testers can give a new example as an input:

"Given, the login page shown,
When, user gives in-valid credentials,
Then, login page is shown again,"
And, "invalid credentials" error shown."

The Security specialist can give a new example to make the system more secure:

> "Given, the login page shown,
> When, user gives in-valid credentials 3 times,
> Then, login page is shown again,"
> And, "Too many attempts. Locked out for 3 minutes" error message is shown,
> And more attempts are not allowed for 3 minutes."

By adding more and more examples, the story becomes more specified. It helps for the teams to deliver the right code for the first time, and to estimate the needed effort better.

The listed examples are the direct inputs for the acceptance criteria and the acceptance tests of the story. Stories and the examples together can massively substitute requirement specification. Examples and acceptance criteria belong not just to stories, but to features or capabilities also.

4.4 The journey of a Capability

Capability is a set of features that makes the product capable to satisfy customer needs. However, usually the set of features are not known at the beginning, only a problem statement and a need for solution. The discussion starts between the customer and a representative in our organization: a sales person, or a business owner if we already have a partnership established with the customer.

Capabilities should be broken up to smaller feature, by considering the Minimum Viable Product and the additional features. When the features are created, the basic examples should be added. This helps to keep the scope focused. After the priorities added to features, and based on historical data, it looks it will part of the next Program Increment, the grooming to stories has to be started. Story level key examples should be added during this activity. We need to have the level of understanding represented by examples, what is enough for starting the PI planning.

PI planning is the point where the teams get the stories in their hands. If they can't commit with the status of stories to the PI, they need to work on it on Team's breakout sessions during PI planning, but the time is limited here. These is most probably not the final list of examples. Further work can be done on Iteration Planning, or on a short re-occurring example mapping workshops.

5 Implementing BDD in the organization

Usually, implementing a new way of working, results in resistance. BDD requires a cultural change. It depends on the company culture, how easily they adopt. A BDD approach affect the daily work of people on multiple levels. So, it is important to handle like a tough organization change.

Figure 9 shows the 8 steps to transforming an organization by John P. Kotter. (Kotter, 1995)

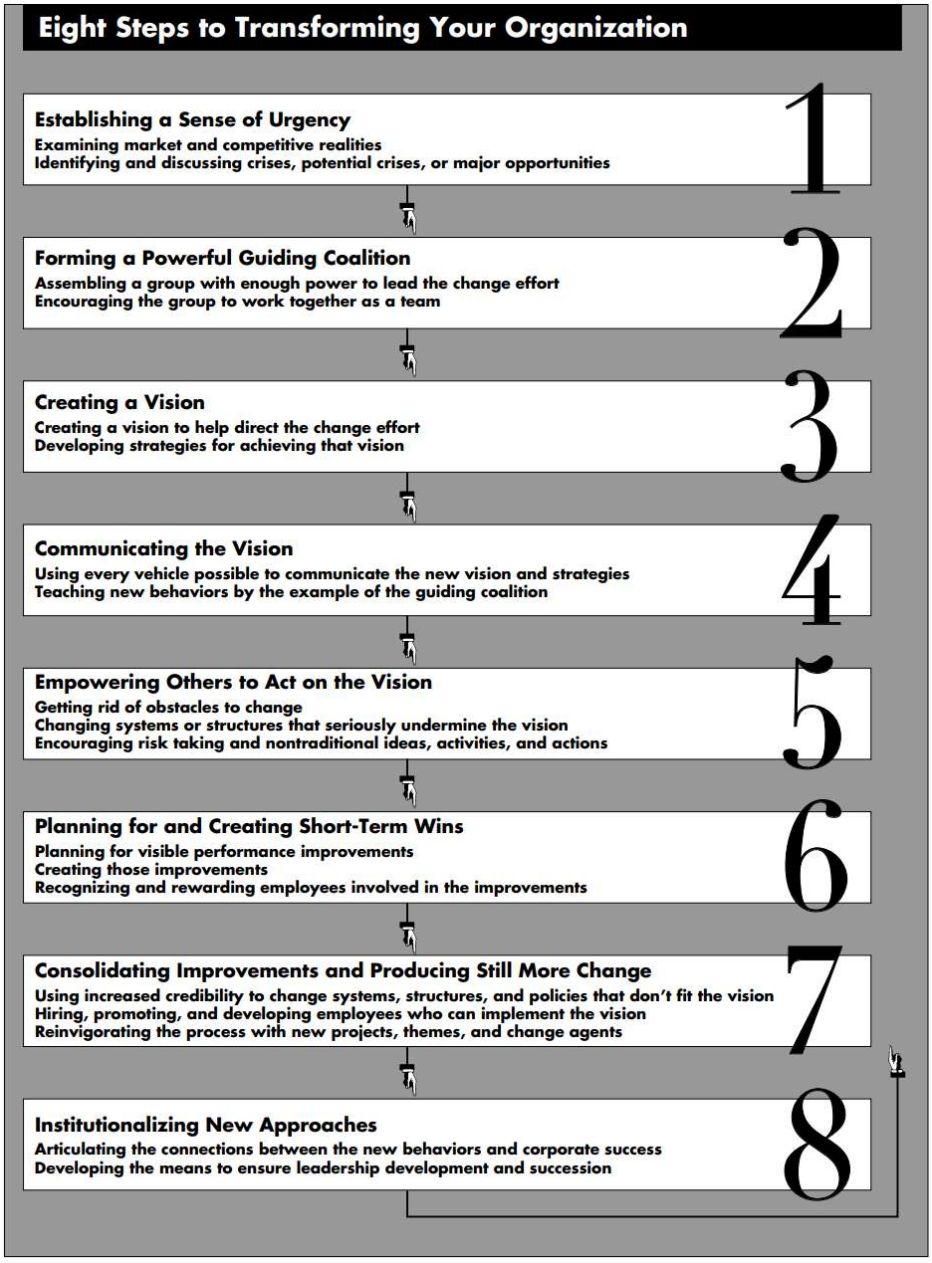

Figure 9: Eight steps to transforming an organization (Kotter, 1995)

5.1 Establishing a sense of urgency

First, we need to understand, what is the benefits of applying BDD. In an overloaded team, it is essential to focus on the valuable tasks, and develop what is needed right for the first time. In many organization, there are several different customer requests in the queue, and there is always a high pressure to satisfy all of it. To be able to do that, the best understanding of business value is essential on team level. Example mapping is serving it, and test-driven development helps to track progress towards the right goals.

5.2 Forming a powerful guiding coalition

BDD requires the collaboration of several functions and areas in development: coding, testing, business, customer relations, documentation, etc… it is a lucky situation if these are set up in a leaned, cross-functional team. Otherwise, it is needed to include key people from different teams, working on the same product. So, the support of implementing BDD is usually required from different team, and because of that, from different hierarchical levels also. But, a top to bottom approach can't be successful either. Engineers also needs to be involved in early phases. At least some experts.

Give time for change! Every change requires effort from the people. In a fully utilized organization change and improve the way of working is impossible.

5.3 Creating a vision

Communicate the purpose of BDD to all stakeholders! Set a set of measures early, before implementing the change! The measurements should be directly connected to your aims to reach by applying BDD. For example: throughput time, number of internal or external fault reports or customers satisfaction.

5.4 Communicating the vision and Empowering others to act on the vision

Find the benefits which are important for the different people! Business will receive faster the product which is closer to their expectations. Development will avoid non-value-added work and will see the direct effect of their work on the customer, and the success of the company

Apply the implementation of the new way of working in small. Make the pilot transparent, measure the success and advertise to other teams! Turn your original assumptions to facts! Use the results of the pilot project to justify the need of change!

BDD directly connects the business with the development. It will be clear how the every-day developments activities create value for customers. By having a wider view overall, employees may take more ownership, and be more motivated.

5.5 Planning for and creating short-term wins

Celebrate every step of success! Work iteratively and adapt for the learning points! Some improvements in measured goals will come slowly, some will come fast. Focus on fast improvements!

Use the BDD examples as acceptance test cases for stories or features. The passing ratio of these test cases are the main progress indicators. When all the acceptance test cases which are included by a story or feature, the developer gets a direct feedback that his work is done. BDD enables the organization to track the progress in small pieces and give the chance to celebrate it.

5.6 Institutionalizing new approaches

Connect the improved company successes with the actions of change! Scale up BDD for the whole organization. Educate other leaders to ensure their development and succession!

Current document writes down a simple, leaned process with clear ceremonies and roles. It can be used as a baseline. Other teams can re-shape it based on their own needs, and the learning points from the first pilots. BDD is a quite straightforward and generic approach to be suitable for different products also.

6 Conclusion

In this thesis I showed how BDD leads the organization to a leaned and more agile operation. I provided a detailed workflow for Behaviour Driven Development of complex products in big enterprises. I detailed the important steps in introduction of BDD and discussed how it can fit in Scaled Agile Framework which is the one of the most popular agile framework at bigger companies.

BDD is a powerful methodology to keep focus on high customer value added features and keep wasted work on a minimum. BDD can be a link to align development with business and improve test automation practices. It is a key factor to improve the agility of the organization.

Scaled Agile INC® also recognized the importance of BDD in agile software development and introduced BDD as a part of SAFe in the 4.6 version, released on 15th of November 2018. This shows the relevance of the topic.

This document provides a good starting point for implementing BDD and see the expected benefits. The next steps are to put this theory into practice. Implement and exercise the theoretical suppositions in a pilot project in practice.

References

Adzic, G. (2011). Specification by example: how successful teams deliver the right software. Manning Publications Co.

Graham, D., Van Veenendaal, E., & Evans, I. (2008). Foundations of software testing: ISTQB certification. Cengage Learning EMEA.

Modig, N., and Åhlström, P., 2013. This is lean: resolving the efficiency paradox. Stockholm: Rheologica Publishing.

Keen, J. M. (2011). Making Technology Investments Profitable: ROI Road Map from Business Case to Value Realization. John Wiley & Sons.

Kotter, J. P. (1995). Leading change: Why transformation efforts fail.

Nagy, G., & Rose, S. (2018). Discovery: Explore behaviour using examples (Volume 1).

Poppendieck, M., & Poppendieck, T. (2003). Lean Software Development: An Agile Toolkit: An Agile Toolkit. Addison-Wesley.

Reinertsen, D.G., 2009. The principles of product development flow: second generation lean product development (Vol. 62). Redondo Beach: Celeritas.

Rose, S., Wynne, M., & Hellesoy, A. (2015). The Cucumber for Java Book: Behaviour-Driven Development for Testers and Developers. Pragmatic Bookshelf.

Scaled Agile, Inc, 2018. Scaled Agile Framework Big Picture [Online], Available at: <https://scaledagileframework.com>, [Accessed 1 September 2018.]

Scaled Agile, Inc, 2018. The SAFe House of Lean [Online], Available at: < https://www.scaledagileframework.com/lean-agile-mindset>, [Accessed 30 November 2018.]

Scaled Agile, Inc, 2018. Weighted Shortest Job First [Online], Available at: < https://www.scaledagileframework.com/wsjf>, [Accessed 30 November 2018.]

Sutherland, J.V., 2014. Scrum: the art of doing twice the work in half the time. First Edition. edn..

Womack, J. P., Womack, J. P., Jones, D. T., & Roos, D. (1990). Machine that changed the world. Simon and Schuster.