Becoming Predictably Adaptable in Software Development

Michael Vakoc, Alena Buchalcevova

University of Economics, Faculty of Informatics and Statistics
Prague, Czech Republic

vakoc.m@gmail.com, alena.buchalcevova@vse.cz

DOI: 10.20470/jsi.v8i4.328

Abstract: *It's difficult to state exact timelines in software development and it is even more difficult to say when features that users want will be delivered. We propose changes to current software development methodologies that enable companies to be predictably adaptable and deliver both on time and what customer asked for. We do so through research of current literature, interviews and personal experience working at an international company that builds products for millions of customers and is facing exactly the challenges described above.*

Keywords: Agile, Scrum, Cynefin framework, software methodologies

1. Introduction

It is human nature to predict what will happen in the future. For centuries, people formed proverbs and predicted weather based on animals' behavior. This practice gave farmers some prediction as to what the yield is going to be like. In software development, we likewise need to predict the future state of the software. The first methodologies evolved together with the first software and they were focused on delivering on time and what the developer wanted. One of the examples is Formal specification (Hoare, 1971) the method was effective in increasing the program quality - by 100 times and return on investment - 33 times according to Mr. McGibbon (McGibbon, 1996). However, there was no mechanism how to capture customer requirements and that became an issue over time.

In this paper we propose modifications to Scrum that enable teams to be both predictable and adaptable. The approach is based on (Bakardzhiev, 2016) where he describes how teams can be predictably adaptable. However, Bakardzhiev (2016) focuses on teams in companies that develop software for customers. Moreover, he doesn't provide any data as to what the results of using the methodology are. We have modified the methodology for teams in companies that develop their own products. We have done so based on review of contemporary literature about software development methodologies, interviews and own experience. The methodology was successfully applied in a big software company. Additionally, we have run tests that provide evidence to the successful application of the methodology.

To explain the proposed methodology thoroughly we first define the terms adaptability and predictability in context of this paper. We then explore the motivation as to why companies and teams should be adaptable in software development, following on to highlight the propositions of being predictable. A substantial part of this work presents a solution to how we can combine both approaches with maximum gains and minimal loses and form predictably adaptable teams. We then present data from applying the proposed methodology and form a discussion about the results.

2. Adaptability and Predictability

Firstly, we would like to explain what we mean by adaptability and predictability in the context of this paper as both terms can have various connotations. To define these key terms, we paraphrase the words of Bakardzhiev (2016).

Adaptability is a measure to which a system can adapt to changing conditions. In the case of software development, it measures how well a team can react to changing customers' requirements, wavering market conditions or results of A/B testing of the product. There are two levels of adaptability:

- Direct adaptation - system acts in a way prescribed by its current set of patterns
- Complex adaptation - system changes or replaces current patterns to create new ones when needed

Predictability is the degree to which a correct prediction of a system's future can be made. In other words, it means how well we can predict the future status of the software at a set time. By status we mean quality and number of features implemented.

2.1 Benefits of Predictability in Software Development

Any successful publicly traded business has some level of forecasting (Lewis, 2016). It is the trust of shareholders in the company's vision that makes the company valuable. The recent example of Uber's valuation of over $50 billion is by big part based on the trust in the vision of the company and its ability to deliver on time. CEOs usually take advantage of mandatory quarterly disclosure of financial statements to predict the market going forward (Microsoft, 2017). Other events, (in the case of Microsoft that is most often Microsoft Build conference) CEOs and product leaders present a vision for the company going beyond just one quarter. For that reason, there is a clear demand for predictability in software development that powers the business. CEOs cannot make predictions without knowing when the features they have announced will be available to the public.

Not delivering on the timings given can cast a negative sight on business leaders that made those predictions – that happened to Tim Cook in 2016 when Apple announced revolutionary AirPods at their special event. The headphones, according to Apple's original announcement, should have been available for purchase in the 2016 Holiday season. However, due to technical complications this timeline was delayed until 2017. This did not majorly impact Apple's sales, but it reflected "badly on Tim Cook and his team" (Ewan, 2016).

Another example of the need for predictability arises from a personal experience. First author was a product manager on a team that was building PSTN calling functionality for Skype and he was working another team was developing a dedicated Android Skype client for the Indian market. The team could not launch the product without the functionality the author's team provided. Given that the launch of the product was a huge event and the marketing department would have to prepare for several weeks before the launch they wanted to agree with author's team on delivery schedule for the feature. However, because the requirement was like none before it was very hard for us to estimate as we didn't have any experience based on which we could provide an estimate. Using some of the techniques outlined in this paper we provided an estimate and the deadlines were successfully set and met.

2.2 Benefits of Adaptability in Software Development

We have highlighted the importance of predictability in software development above. In this section we outline the benefits of adaptability. The agile methodologies have brought the focused principle of listening to the customer and shipping the features they ask for as soon as possible. New requirements may (and often do) arise, as a result of customer's feedback about delivered functionality

Regarding the Skype example mentioned above if we had taken the path of pure predictability and made an elaborate estimate on how much work would it require we could with a bit of luck predict when the feature will be out. However, what if in the middle of the development, colleagues from the supported team realized they needed the feature to behave like B in one scenario and not like A as previously planned because of results of some early stage user tests? We would have been stuck with our plan, would have to finish it (on time) and then bend the functionality to meet the need. This could take a long time and prolong the whole process of shipping features that customers want to see on the market.

Had we focused on the pure agile approach and built the feature with no planning and were ready to include any changes in the middle of the development, we might finish the feature almost on time. Yet if this approach was followed we would not have an original deadline estimate and the marketing team and Android client team would not have any date to which they could refer to and strive to achieve.

2.3 Motivation for Being Predictably Adaptable

The approaches presented in sections above both have positives and negatives. Based on the context of the project, predictability might be preferred over adaptability or vice versa. As stated in the introduction, the goal of this paper is to show the benefits of being predictably adaptable and how to achieve this. In this section we present the motivation in being predictably adaptable in further detail, giving an example of such an approach.

A great example of the predictably adaptable approach was the development cycle of iOS (operating system that powers mobile devices from Apple) a few months after the iPhone was originally introduced. The phone launched in June 2007 was a success straight away – 270,000 units were sold during the first weekend of sales despite the high price tag (Reuters, 2008). The first iPhone launched with applications developed solely by Apple. It wasn't until January 2008 when Apple introduced the App Store – a marketplace that allowed iPhone users to install apps from 3rd party developers. Apple showed great prediction with launching the iPhone and making the content the most important element of the phone. It wasn't until later that Apple realized the company itself could not supply all the great applications people wanted. If they didn't open the ecosystem through App Store and allow 3rd party integrations, iPhones might have never been as successful as they are today– thus the importance of adaptability (listening to customer needs) shows as very important.

There are learning for everyone. If you believe you have the best product and everybody will love it, you still might have to change direction after launch. If you choose to be blind to the feedback from customers you will probably never make it (Laugusen, 2010).

2.4 Being Predictably Adaptable with Scrum

Scrum is the most popular development methodology now (ScrumAlliance, 2013). Gurendo (Gurendo, 2015) describes Scrum in five phases

- Product backlog creation
- Sprint planning and sprint backlog creation
- Working on the Sprint. Scrum meetings
- Testing and product demonstration
- Retrospective and next sprint planning

Each of the Scrum phases is important. However, in this work we will focus on phase 2 – Sprint planning and sprint backlog creation.

In this chapter we have described the benefits of being predictably adaptable. The central argument of this work is that by finishing Scrum sprints successfully, team and company become predictably adaptable. There are two main reasons for that:

- Product owner can predict how much work will be done in the next sprint.
- The team is agile enough to take changed/new requirements into next sprint.

These two points are enough for product owner to have confidence in both setting deadlines and aligning with the latest feedback from customers.

2.5 Software Product Businesses and Software Service Businesses

Cusumano (2003) and Nambisan (2001) divide software companies into service and product companies according to the way they conduct business.

- Software product businesses create products that they then sell to other companies.
- Software service businesses provide mainly consulting related to software development.

The distinction above is important for the next part of the paper.

3. Methodology for Being Predictably Adaptable

Bakardzhiev (2016) proposes a powerful approach on how to be predictably adaptable. However, his methodology is focused on teams in software service businesses and is not well applicable for teams in software product businesses. This distinction is important as the stakeholders are different in each of those types of businesses and certain processes are impractical to copy from one environment to the other.

We propose modifications to current development methodologies that are applicable for teams in software product businesses We have done so based on interviews with engineering teams, study of contemporary literature (as referenced in the text) and authors' own experience. We have purposefully focused on introducing as few new aspects to the methodology as possible as one of the most common problems when adopting new methodologies is that developers see the processes as unnecessary overhead (Freeman, 2015).

The methodology as described by Bakardzhiev (2016) is focusing on contextualization sessions in software service businesses. Bakardzhiev (2016) argues that by having two contextualization sessions an agreement on the complexity of tasks will emerge. However, it is impractical to have two contextualization sessions teams in software product businesses as we will demonstrate in the coming chapters. We will describe the differences and we will report on results from testing the methodology with real world teams.

3.1 The Proposed Methodology

Companies need to move faster than ever (Fridman, 2015) and to do so changes to current software development methodologies are needed.

In chapter 2.4 we have listed the five phases of Scrum. The changes that we propose are impacting phase 2 – Sprint planning and sprint backlog creation. Phase two consist of sprint grooming, planning and the creation of the sprint's backlog. During the grooming session, estimations are created using the silent estimation.

We propose changes to how majority of the companies handle grooming. Like Bakardzhiev (2016) we introduce a contextualization session that helps the team understand better the complexity of tasks and so as a result make better estimates. As mentioned in previous chapters, Bakardzhiev (2016) focuses on software service businesses, the changes proposed in this paper are aimed at software product businesses instead.

During contextualization we apply Cynefin framework to "groom" backlog items within the development team. This session is called Contextualization as it was first named by Dave Snowden.

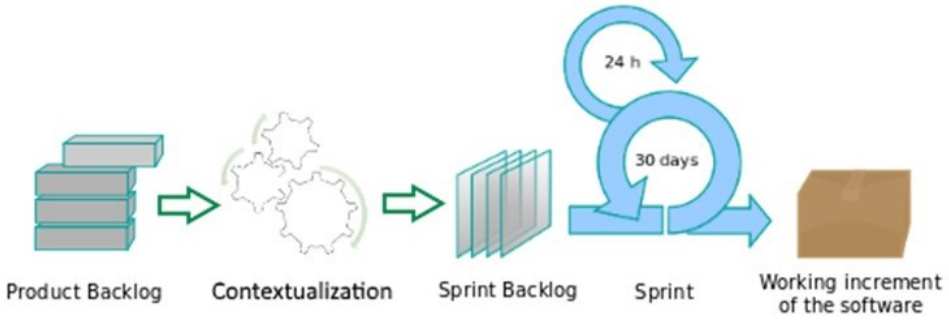

Figure 1: Contextualization session in context of Scrum process.
Source: http://www.yodiz.com/blog/wp-content/uploads/2016/01/Scrum_process.svg_.png

3.1.1 Cynefin Framework

Before we describe the details of contextualization session, we present the Cynefin framework (Snowden, 2007) that has been developed to help deal with complex systems and was originally developed to help leaders make better decisions by better understanding the operating context.

Snowden (2007) says that: "Cynefin framework enables to see things from new viewpoints, assimilate complex concepts, and address real-world problems and opportunities. Using this approach, leaders can learn to define the framework with examples from their own's organization's history and scenarios of its possible future. This enhances communication and helps executives rapidly understand the context in which they are operating"

We argue that the statement is true for product managers and product owners too, since they are in fact, leaders of the development teams.

Cynefin framework consists of five domains;

- Obvious
- Complicated
- Complex
- Chaotic
- Disorder

Each of these domains require different set of actions. Obvious and complicated domains assume an ordered universe where cause and effect is visible. Complex and chaotic domains expect unordered universe where case and effect are not connected in any apparent way.

We argue that the same logic could be used for task estimation. When a task fits into order it should be estimated based on facts about the task. However, if task falls into unordered universe, it requires pattern based estimation where we should look for patterns that we previously have experience with.

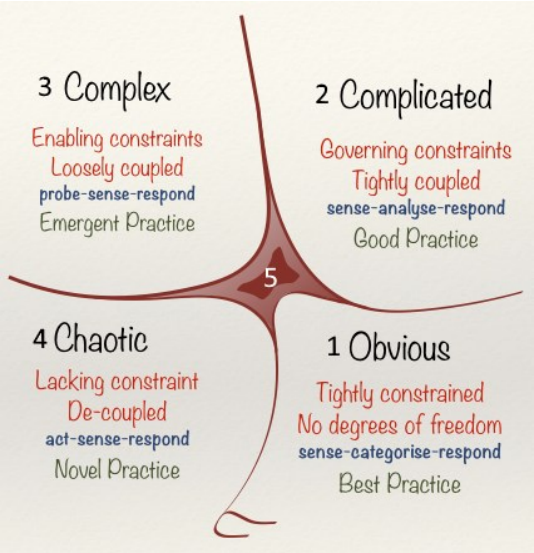

Figure 2: By Snowden, *Source:* https://commons.wikimedia.org/w/index.php?curid=33783436

3.1.2 Contextualization

As part of the contextualization, the development team silently votes which domain of Cynefin framework they think the individual task belongs to. This enables us to determine the overall understanding of the task in the team.

If there is disparity between which domain the members of the team assign the task to, communication is encouraged to ensure that everyone has the same understanding of the task.

Because of Bakardzhiev (2016) is focused on software service businesses he suggests that two contextualization sessions are hold one with the customer, the other with the development team. The customer round session is intended to determine clarity of requirements and the second round to determine the capabilities of the team to fulfill those requirements. Ultimately a categorization matrix is used to define item's uncertainty level.

Having two contextualization sessions is a lot of overhead in the case of a team in software products company. Having one more session might prevent the greatly designed methodology from being implemented by the team (Montoya, 2014). Therefore, we propose to have only one contextualization session.

In traditional grooming in Scrum, each of the developers express how many story-points they think the item is worth based on their knowledge, previous experience etc. However, not enough is often made to make sure that everyone has the same understanding of the problem and so estimates may vary greatly.

Bakardzhiev (2016) suggests that the development team first focuses on clarity of requirements and common understanding of the problem. If the product owner thinks that the item is in a different domain than developers she needs to explain the requirement clearer and make additional changes to the backlog item to highlight the problematic parts. We argue that the estimation process is then more precise and story points given to items represent the reality better. In next chapter of this paper we show data to support this argument.

One of the authors has tested the proposed activity in practice. In chapter 5 we present results of introducing it to two development teams.

3.2 Benefits of proposed methodology

One of the main benefits of the proposed approach over traditional grooming in Scrum is that the whole team acquires the same understanding of complexity included in each of the product backlog items. This in turn enables the team to make more precise estimations which leads to more successful sprints and the team will become predictably adaptable.

The benefits over the methodology Bakardzhiev (2016) suggests is that it focuses on software product companies.

4. Methodology Testing

After we have defined the methodology, we have tested whether the methodology really helps teams make the sprints more successful - a key element in making company predictably adaptable as we suggested in the second chapter of this paper.

First author had access to data about 2 teams in a big international company for past 20 sprints and the author has set the period for making reasonable conclusions after approximately one third of this time - 7 sprints for collecting data using the updated methodology. During this period, the author has focused on one thing - success rate of sprints conducted by the teams.

Let us first define what was considered a successful sprint. Most of modern resources consider sprint a success if the definition of done was met for all items (Scruminc, 2017). This includes the development of the code, testing it, and then being accepted by product owner during the sprint's demo.

Data from 20 sprints for two different teams:

			Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5		Sprint 6		Sprint 7		Sprint 8		Sprint 9		Sprint 10	
	success	Size	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R
Team 1	20%	10	35	31	33	28	34	22	33	33	33	31	30	25	30	30	32	30	30	18	28	24
Team 2	65%	12	42	42	45	40	42	42	44	44	44	40	42	40	40	37	40	40	42	37	42	44
Team 1 (cont)			Sprint 11		Sprint 12		Sprint 13		Sprint 14		Sprint 15		Sprint 16		Sprint 17		Sprint 18		Sprint 19		Sprint 20	
Team 2 (cont)			P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R
			29	29	30	28	30	22	26	24	28	20	24	20	28	28	31	30	30	24	27	25
			44	44	44	41	42	42	42	44	44	44	42	42	44	44	44	40	42	42	42	42

Figure 2: 20 sprints in data. P = Planned, R = Reality, red means that a sprint was unsuccessful - more story points committed to then actually delivered, green means sprint was successful.
Source: author

- Team 1: 20% success rate
  - size: 10 engineers located in the same location, product owner remote
- Team 2: 65% successful
  - size: 12 engineers located in the same location, product owner remote

From the data above, we can see that the teams are similar in composition and the way they work with product owner. However, they differ substantially in how successful the sprints are. One team has extremely low success rate of the sprint (20%) whereas the second one can be considered moderately successful. (65%) The teams were chosen deliberately because of their similarities and yet different success rates of their sprints. This specific data sets helps to prove or disprove the methodology as to whether it can help both successful and unsuccessful teams to achieve better results.

The methodology was applied in both teams and the results were measured for 7 two-weeks sprints. We discuss the results in the next chapter.

- Team 1: 57% success rate
- Team 2: 71% success rate

	success	Size	Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5		Sprint 6		Sprint 7	
Team 1	57%	10	28	28	28	26	30	31	30	25	28	28	30	30	31	27
Team 2	71%	12	44	44	48	48	42	40	41	44	42	40	44	44	44	46

Figure 3: Data from 7 sprints after implementing changes to Scrum as suggested in this paper

5. Discussion

The proposed methodology is based on Cynefin framework and subsequent contextualization which has been suggested for use in software development by Bakardzhiev (2016).

The basic predicate for the paper is that successful sprints mean that a company is predictably adaptable. In chapter 2.4 we present the reasons why. Furthermore, we argue that predictability is important for alignments in the company. The importance of these alignments was described at the beginning of this paper where we have illustrated predictability on two examples.

There are two main contributions of the paper (i) description of enhanced grooming techniques that build on Bakardzhiev (2016) and (ii) testing the methodology in practice.

The proposed changes to Scrum's grooming were tested in a multinational company (teams were distributed across multiple time zones) and are enough to conclude that the modifications have benefits over the grooming techniques that was used by those teams previously. Substantial improvement was measured especially in the team where success rate was low before applying the changes – Team 1. (Sprint success rate improved by 37 percent points). Slight improvement was also noticeable in Team 2 (6 percent points).

Based on the data presented in chapter 5 we can see that the biggest problem that prevented Team 1 from being successful were wrong estimations. The changes we proposed helped the team make better estimates significantly. For the already successful team (Team 2) the success rate has also improved but additional testing would have been needed to claim that the methodology clearly helps even to already successful teams.

Moreover, further testing is needed to prove that the methodology works in small and medium sized companies as the data above was collected in a big multinational company.

6. Conclusion

It is not enough for companies to be agile. Companies need to be predictably adaptable and to do so, software development methodologies need to evolve. We have proposed changes to Scrum which are based on Bakardzhiev (2016) and modified it to work for teams in Software products companies. We have then carried out tests which proved that the methodology helps teams become predictably adaptable.

The methodology described in the paper has proven potential to help especially teams that currently have low success rate for their sprints. Further testing needs to be done to proclaim its success also in small and medium businesses and for teams in big companies that already have majority of sprints successful.

References

Bakardzhiev, D., 2016: *Adaptable or Predictable? Strive for Both – Be Predictably Adaptable!.* [Online] Available at: https://www.infoq.com/articles/predictably-adaptable?utm_source=infoqWeeklyNewsletter&utm_medium=WeeklyNL_EditorialContent_culture-methods&utm_campaign=09132016news [Accessed 22 10 2016]

Basili, V. R. &. T. J., 1975: *Iterative enhancement: A practical technique for software development.* s.l.:s.n.

Cusumano, M., 2003: Finding Your balance in the Products and Service Debate. *Communications of the ACM,* 46(3): 15-17

Ewan, S., 2016: *Forbes.* [Online] Available at: https://www.forbes.com/sites/ewanspence/2016/11/01/apple-iphone-airpods-delay/#1ea6b97d1d82 [Accessed 10 July 2017]

Gurendo, D., 2015: *XB Software Blog.* [Online] Available at: https://xbsoftware.com/blog/software-development-life-cycle-sdlc-scrum-step-step/ [Accessed 27 09 2017]

Hoare, C., 1971: In: *Proof of a program: FIND. Communications of the ACM.* s.l.:s.n., pp. 39-45.

Laugesen, Y. Y., 2010: *What factors contributed to the success of Apple's iPhone?.* Ontario, s.n.

Lewis, R., 2016: *InfoQ.* [Online] Available at: https://www.infoq.com/articles/predictable-agile-delivery?utm_campaign=rightbar_v2&utm_source=infoq&utm_medium=articles_link&utm_content=link_text [Accessed 13 11 2016]

McGibbon, T., 1996: *A business case for software process improvement.* Rome, NY: s.n.

Microsoft, 2017: *Investor Relations.* [Online] Available at: https://www.microsoft.com/en-us/Investor/earnings/FY-2017-Q3/press-release-webcast [Accessed 10 July 2017]

Montoya, M., 2014: *CPrime.* [Online] Available at: https://www.cprime.com/2014/03/the-3-biggest-challenges-of-adopting-scrum/ [Accessed 13 11 2016]

Reuters, 2008: *New York Times.* [Online] Available at: http://www.nytimes.com/2008/07/15/technology/15apple.html [Accessed 1 December 2016].

Nambisan, S., 2001: Why Service Business are not Product Businesses. *MIT Sloan Management Review,* 42(4): 72-80

ScrumAlliance, 2013: *ScrumAlliance.* [Online] Available at: http://www.scrumalliance.org/scrum/media/ScrumAllianceMedia/Files%20and%20PDFs/State%20of%20Scrum/2013-State-of-Scrum-Report_062713_final.pdf [Accessed 13 11 2016].

Scruminc, 2015: *Scrumin.* [Online] Available at: https://www.scruminc.com/definition-of-done/ [Accessed 1 June 2017]

Snowden, D., 2007: *A Leader's Framework for Decision Making.* [Online] Available at: https://hbr.org/2007/11/a-leaders-framework-for-decision-making [Accessed 27 6 2017]

JEL Classification: L86, M15