Feature prioritization in SAFe model using COCOMO II

Hessam Emami
47689516
hemami@smu.edu

Abstract

Software requirements prioritization is considered as one of the significant challenges for software managers in today's IT business. Depends on the software development model, the requirements ranking technique is different. This paper presents a case study to demonstrate the accuracy and practicality of using COCOMO II in the SAFe model to estimate the effort that is required to develop new features. The effort can be used for what SAFe recognizes as size in Weighted Shortest Job First (WSJF) model to prioritize the work for each Program Increment (PI).

Keywords— Effort Estimation, SAFe, Feature prioritization, COCOMO II

1. Introduction

Agile is a time-boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end [1]. In small companies, Scrum as an Agile process framework is used to manage teams and deliver software every 2-4 weeks[16]. However, in a large organization that needs multiple dedicated teams to work on a project, additional effort is required to coordinate the collaboration among the teams, sync development, releases and activities, and, most importantly, manage multiple sources of requirements.

SAFe is the world's leading framework for scaling Agile across the enterprise. Used by hundreds of the world's largest organizations, SAFe sustains and drives faster time-to-market, dramatic increases in productivity and quality, and improvement in employee engagement [2]. Depends on the size of the organization, SAFe provides different solutions, and in this paper, Essential SAFe that includes Scrum and Program teams is considered.

To handle requirement priorities, SAFe uses Weighted Shortest Job First (WSJF) model:

WSJF = Cost of Delay / Job Duration (Size)

Determining the Job Size is different from one organization to another, and There are many heuristic approaches for agile estimation, but planning poker and estimation by Analogy are the most popular approaches in the industry. COCOMO II is suitable for traditional development processes such as waterfall; however, there are some researches like [3] try to incorporate the effort and cost formulas for the Agile method. As these methods introduce new cost drivers that are not calibrated in industry, the effectiveness of the methods is not proven. According to [4], the refined COCOMO III proposal will not consider Agile as well; therefore, the effectiveness of COCOMO methods for project cost and effort assessment in Agile is under question.

SAFe model presents a Program Increment (PI) approach that some argue it partially mimics the waterfall methodology. Although there are significant differences between these two, 3 months of advance planning is not entirely compatible with the Agile concepts. In this paper, we present a case study using COCOMO II cost drivers and scale factors to estimate the effort of past features as the size in the WSJF formula and compare the outcome of the actual result to what COCOMO II could suggest if it has been used for prioritization. Also, we will propose a cost driver that can assist the better estimation for effort in big organizations that use the SAFe Model.

2. Background and related work

In traditional project management, determining a process that can clarify project objectives and deliverables, show the scope of the work, control risks, and manage resources are the primary goals of the project manager. Depends the organization structure is closer on which side of the matrix management spectrum, some of the responsibilities can be shared among the other units, but regardless of dealing with a strong matrix, week matrix, or something between these two ends, the objective is to establish this process first.

On the other hand, Agile is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment [4]. Since the Agile Manifesto was published in 2001, the number of organizations that have been trying to adopt agile methods as a way to deal with software development has increased [4]. During about two decades since Agile manifesto released different frameworks and methodologies such as Scrum, Feature-Driven Development, and Extreme Programming emerged to demonstrate how Agile can be implemented and replace traditional project management in a company. However, many of them have not reached their adoption goals, which include fast and high-quality software deliveries, software products that better satisfy users' needs and flexibility to deal with scope changes throughout the project [5]. Large enterprises suffer more than a startup or small company for applying Agile methodologies in their process. Using Scrum, while keeping an eye on the strategy and the goals of the overall enterprise, is not easy.[7] Often teams lose sight of these objectives and try to achieve a project-specific rather than an enterprise-wide optimum [7]. There is a competition between execution teams that attempt to deliver the work to production as fast as possible and architects that they aim to deliver a solution that can be integrated into enterprise infrastructure and does not impose any risk for maintaining and supporting it in the future. Extensive collaboration among teams is required in an enterprise to develop and release deliverables to production, and in most cases, project dependencies delay the releases. Many believed Agile could not be fully functional in large enterprises due to complicated bureaucratic systems, regulations, dependencies, and resource management. In their opinion, problems are too complicated in an enterprise that without having a restricted and well-defined process be manageable, but also it is impossible to neglect the benefits of using Agile.

Scaled Agile Framework (SAFe) is an attempt to implement a scalable Agile in an enterprise. Scaled Agile Framework (SAFe) empowers complex organizations to achieve the benefits of Lean-Agile software and systems development at scale [2]. Although SAFe provides guidelines to implement Agile still, each enterprise has the flexibility to adjust the processes and flows and bring their perceptions about some definitions if it is required. During the planning phase, it could be a competition among customers to prioritize their work for the upcoming Program Increments (PI). SAFe uses the WSJF method to sort the feature's priority; however, it does not define the feature size precisely. Job size in SAFe mostly refers to the job duration meaning how long it takes to develop and deliver the job to production. This definition of the size requires teams to know in advance about the high-level design of the feature, the potential codebase size, technology dependencies, platform readiness, team skills, and so many other factors. Sizing methods like T-shirt sizing proved that it could not be accurate specifically for large features. Some delegates from a team that will be responsible for delivering a feature cannot consider all of the mentioned parameters and give an estimate that can be practiced as size in the WSFJ strategy. There is another major difficulty with the definition of size in the SAFe model. In Agile methods, story points are mostly considered to determine the size of the job, not job duration. [13] indicates three notable problems that exist in Agile estimation. First, it cannot be easily related to the time duration because story points represent the amount of work, and the velocity differs from team to team [13]. Second, because a story point is a relative value, the total story point value can fluctuate with a slight variation in the baseline story point [13]. Third, after the story points and velocity are used to estimate the initial size of the project, a schedule is derived using a simple linear extrapolation [13]. Therefore, despite SAFe is an Agile method, job duration as the project size cannot be measured by conventional Agile estimation methods.

Procedural software cost estimation methods are not fully aligned with Agile, but they seemed as a conforming practice to determine the feature size in the SAFe model. The leading method that uses the model approach for estimation is COCOMO-II. COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and is developed at the University of Southern California. It is the model that allows one to estimate the cost, effort, and schedule when planning a new software development activity [14]. Some studies try to utilize COCOMO-II in Agile. Studies like [3] apply COCOMO-II for agile and even introduce a new algorithmic estimation method to fill the gap for project size estimation using this method. The concern using their method for this study is that they only focus on a small team that can follow all Agile best practices during the project process. By knowing the structure of the SAFe, that consideration cannot be compatible with the study use case. Therefore, in this study, the original COCOMO-II method will be used for some past projects, and the author introduces a new cost driver that can help to measure the job size more accurately.

3. COCOMO-II for estimation

In this paper, a large enterprise company with several organizations and lines of business is considered. Each

organization includes different LOBs which its objectives and management system may be different from each other. For a specific LOB, teams are formed under the SAFe framework, and they are managed in an Agile train. Essential SAFe that has Scrum teams and the program layer is the type of SAFe framework that this train uses. The team level of the framework consists of agile teams, which are collectively responsible for defining, building and testing software in fixed-length iteration and releases.[10] There are dedicated and temporary Scrum teams on this train. Every three months in a three days program increment planning meeting, teams pick up the prioritized features and move them into their backlog. Based on the feature's size, team capacity, and skills, Scrum teams would determine delivery time for each feature. At the program level, the goal is to provide a healthy backlog for the train in the form of epics for features that they can be distributed among Scrum teams in the planning meeting later. Product owners, release engineers, and architects work together to investigate the work priorities and technical feasibility of the requirements. Another responsibility for the program team is to define the business roadmap that is compatible with the portfolio roadmap. The portfolio level is needed for large enterprises, which require governance and management models. The essence lies in achieving a balance between four potentially conflicting goals [12]:

- maximizing the financial value of the portfolio [12] by identifying value streams using Kanban systems [10],
- linking the portfolio to the strategy [12] of an organization through investment themes [10],
- ensuring that the scope of activities is feasible [12] by measuring appropriate metrics [10], and
- balancing the portfolio on relevant dimensions [12] by defining and managing business and architectural epics, which run across value streams [10].

Enterprise companies require portfolio level, but, in this study, we do not deal with the portfolio roles in work prioritization because it desires, a long-term study, and familiarity of vital business objectives, and it does not have to determine the priorities in the short term.

Statistical data from three previous PIs shows WSFJ has a poor performance for prioritizing the work. Table [1] shows the estimated required iterations and the actual results for the first 4 high priority features in this train:

Table 1 Feature - Estimation

	Feature A	Feature B	Feature C	Feature D
Estimated (iteration)	4	3	2	5
Delivered (iteration)	Carry on	10	Blocked	3

Each iteration in this train takes two weeks, and as table[1] shows in the last 3 PIs that includes 19 iterations (1 PI had 7 iterations) 25 percent of the high priority features could not be delivered, one feature is in blocked state and the actual size of one of them (Feature B)is three times more than what it was estimated. On the other hand, Feature D took two iterations less than what was predicted. In this train, Scrum teams are considered as the executive teams that are responsible for delivering the work, and the program team is responsible for maintaining a healthy backlog at least for one PI in advance. The program team consists of two teams. The business personnel who their leading role is to create the requirements and IT team, which is responsible for investigating the technical feasibility of the requirements, negotiating on delivery schedules, and providing a size estimation for each feature. After features delivered to Scrum teams, they will break them down into story level, use story points, and their team velocity to estimate the delivery time. The problems that were observed in this method are first the influence of the program team size estimation on the Scrum team points, although the program team is not responsible for delivering the feature. Second, prioritization happens before Scrum teams break down the work, so it is not clear for some specific parameters such as team skills how the program team assesses them. One possible solution is to ask some delegates from Scrum teams to participate in the sizing exercise to provide a better estimate for each feature. However, this approach also does not guarantee that the final estimation is correct. Depending on the type of the feature, this solution needs delegates from all Scrum teams as each team has different skill sets and velocity, which leads to consuming Scrum teams time for planning and requirement management. Considering that in each PI teams plan for the next three months in advance for the requirements that are defined in the backlog, the program team should have a solid idea about the feasibility of each requirement also as they have been working with Scrum teams and precisely tech leads of each team, they should have a good understanding about train readiness to accept the work. If we look at the assumption from a higher level, a mathematical

equation could be able to deliver a more precise estimate for the size. COCOMO-II by applying scale factor and cost drivers can facilitate the estimation effort.

COCOMO-II has a notable dependency on historical data, so it will be useful if it is used for the entire train for several years, so cost drivers and scale factors can be adjusted accordingly. This enterprise does not have the historical data; however, there is an acceptable measuring data that can be used to collect the best values for the cost drivers and scale factors. In this study, we employ those measurements and utilizing expert judgment to choose the possible base values for each parameter. It is necessary to mention that each PI has more than four features in the program team backlog. However, we will only consider the first high priority features in this paper because first, these features are independent of each other as each one belongs to an independent experience. Secondly, we will consider appropriate values for some factors like process maturity and schedule in order to make sure the impact of the other features in size are included. Also, because some of the features are extra capabilities for the current applications, and for the other ones, the program team provided a high-level design, COCOMO-II post architecture is used for size estimation. To create a baseline for the cost of delay, we use the following table to convert categories to numbers:

Table 2 Requirement delay cost category

Requirement delay cost category	Requirement delay cost numeric value
Critical	10
Very important	7
Important	5
Useful	3
Others	1

Table [2] shows the business cost group of each feature:

Table 3 Feature - Cost group

	Feature A	Feature B	Feature C	Feature D
Cost group	Critical	Critical	Important	Useful

Table [3] shows features priorities based on the WSJF system calculated by using the estimate job size:

Table 4 Feature - WSFJ

	Feature A	Feature B	Feature C	Feature D
WSJF	2.5	3	2.5	0.6

But if we consider the actual job size, we will have table [4] for WSFJ values:

Table 5 Feature WSFJ – actual job size

	Feature A	Feature B	Feature C	Feature D
WSJF	N/A	1	N/A	1

As table [5] shows choosing Feature A and Feature C as high priorities features are not the best options in his case. Also, we can see the WSFJ value of Feature D now is the same as Feature B, so their priority should be considered the same.

Using COCOMO-II size estimation requires finding the right values for cost factors, scale factors, also the number of lines of code required for developing each feature. Table [6] and table [7] show the scale factors and cost drivers for these features. For the scale factors and cost drivers, we have a dedicated column for each feature.

Table 6 Feature - Scale factors

	PREC	FLEX	RESL	TEAM	PMAT
Feature A	N	N	H	N	L
Feature B	N	N	H	N	L
Feature C	N	N	N	N	L
Feature D	H	N	N	H	L

We could consider scale factors for all the features the same and, consider the values for the whole train rather than each feature. However, as the program team has some ideas and predication that which team will be responsible for each feature, we decided to calculate the effort more accurately. Table [6] show the cost drivers for each project.

Table 7 Feature - Cost drivers

	Feature A	Feature B	Feature C	Feature D
RELY	L	L	L	L
DATA	VH	VH	VH	VH
CPLX	N	N	L	L
RSUE	L	L	L	L
DOCU	N	N	N	N

TIME	N	N	N	N
STOR	N	N	N	N
PVOL	N	N	N	N
ACAP	N	N	N	N
PCAP	H	N	N	N
PCON	H	N	N	N
APEX	N	N	N	N
PLEX	N	N	N	N
LTEX	N	N	N	L
TOOL	H	H	H	H
SITE	L	L	L	L
SCED	N	N	N	N

One additional input data is required to be able to calculate the effort. We need to calculate KSLOC for each project. Study the features that have already implemented gives us this advantage to calculate the size accurately. Although, Feature A is a carryover feature and Feature C is blocked due to dependencies still we can retrieve an estimated value based on the current project code size.

Table 8 Feature - SLOC

	Feature A	Feature B	Feature C	Feature D
SLOC	2K	6K	500	200

Table [8] only shows the size of the code that is required to be developed by Scrum teams. We ignored the auto-generated code and CI/CD configurations. The majority of the build and delivery pipeline is handled by the infrastructure and configuration portion of the code can be ignored.

COCOM-II schedule estimation for each project is calculated as follows:

Table 9 Feature estimate

Feature A	6 months
Feature B	5 months
Feature C	4 months
Feature D	3 months

As table [9] shows, there is a significant discrepancy between what the program team estimated for the project duration and what COCOMO-II calculated. If we look at the estimation for Feature B and Feature D, we can find out that the estimation for Feature B is very close to what the actual results table shows. COCOMO-II accurately recognized that Feature D needs less development time than Feature B, something that the program team was not able to estimate correctly; however, the estimation still is far more than what was required. For Feature C, COCOMO-II was not able to consider the dependency correctly. The first assumption could be some cost drivers need be adjusted, and the value of the cost drivers are not accurate. However, Analyst capability and platform volatility, two closest cost drivers that can address this issue, are not quite suitable for dependencies. The type of dependency that blocks Feature C is external, and it is almost out of the control of the train. The counter-argument may suggest a team of sophisticated analyzers should predict this type of impediment in the project. Time to develop for Feature A is more accurate than the program team estimation, but after about 9 months, still, the requirement is not delivered, which means the schedule provided by COCOMO-II also is not acceptable.

3. Introducing a new cost driver

Table [9] shows that COCOMO-II can provide a more accurate estimation for the project size, but the result is not entirely reliable. We found out one project stays blocked, and another one takes more time to be deployed. The question is if COCOMO-II estimation approach can solve the problem, or we need to make some adjustments to this technique. First, we need to find the root cause of the problems. In fact, the wrong estimation for all three Features of A, C, and D stems from one reason.

The environment that each of these applications needs is not ready for development teams. Some cost factors in COCOM-II indirectly consider some aspects of the application environment, but none of them explicitly refers to this issue. For example, complexity is divided into five areas: Control operations, computational operations, device-dependent operations, data management operations, and user interface management operations [15], but all of these factors only consider the complexity of the operation not level of dependencies on external systems. Even the number of input or output dependencies cannot be introduced as a reliable measurement. Also, platform cost factors such as platform volatility or analyst capability are not directly addressing the issue. Software tool cost as another cost driver considers development tools, not what application is required. Storage is the only cost driver that addresses this problem partially. Having a dependency on storage is one of the environmental factors that can impact the delivery of the project. Considering the cost and availability of storage, the rating levels of this cost driver do not match with today's software requirements. Including storage in

estimation could be valid, but instead of a rating system based on usage percentage, it should be based on data security, response time, and regulation compliance. Without considering application environment requirements in an enterprise, any estimate cannot be reliable. COCOMO-II considers architectural risks as a scale factor for the entire project. On the surface, it seems this scale factor can be an excellent candidate to change the project size based on architectural risk, but as the data shows, it could not always predict the problem. The reason for that is, architectural risk mostly refers to high-level design risks that can be an impediment to deliver the feature. As we can see, Feature C was not architecturally significant, and that value of RESL is nominal. Because scale factors applied to an entire feature, it may ignore the low-level design obstacles. In a waterfall, that may not be an issue, but we need a cost driver that can be applied to low-level design modules, which is something that architects cannot consider, to achieve a better size estimation. Agile gives teams the flexibility to choose the technology that can shape the low-level design. If low-level design technology causes legal or technical issues in one or some modules, high-level estimates can be changed.

There are multiple solutions to mitigate this issue:

1) Defining new cost drivers that can include the impact of applications environment such as continuous integration/ continuous delivery readiness, interface readiness, storage readiness.
2) Consolidate the above cost drivers into one and introduce environment readiness as a new cost driver

The first option potentially can provide a more accurate estimation if Scrum teams receive the low-level design in addition to high-level design from the program team. For some specific project, this might be the desired solution, but in the Agile methodology, the execution team should have the power to make decisions for the technologies that they want to use. If we only consider one cost driver as an environmental readiness, the program team does not need to specify the technology that needs to be used for each component. Instead, their responsibility is only to make sure there is a way to implement the feature. The program team can measure the hardship of implementing the feature and pass the information to Scrum teams. If multiple options are available, Scrum teams can adjust the estimate, but both execution and management teams know, in the end, the feature will be delivered.

Table [10] shows the details of this cost driver. Environmental readiness indicates at what level the enterprise application environment is ready for the feature modules. This includes all the factor that is required to deliver the requirement for both test and production environments for each module. In some cases, as we will see for Feature D, this cost driver can facilitate the delivery of the feature and reduce the time to delivery.

Table 10 Environment readiness values

XL	1.74
VL	1.34
L	1.17
N	1
H	0.87
VH	0.73
XH	-

In this paper, the value of this table is set similar to the complexity value of the COCOMO-II. Adjusting the value of this cost driver requires studying several historical data from different companies and cannot be achieved in a short period. However, by examining this new cost driver, we should expect the final results are more accurate than the original scheduling estimation.

4. Utilizing the environment readiness

We are going to apply the new cost driver that we defined in the previous section to estimate time to delivery for discussed features. We need to pay attention to this fact that storage cost driver is considered in environment readiness cost driver. In our case, a storage cost driver is a nominal value, so it does not impact the result, but if in a project, the value is different it is recommended to ignore it to eliminate duplicated calculation.

Table 11 Feature - environment readiness

Feature A	XL
Feature B	L
Feature C	XL
Feature D	VH

Table [11] demonstrates the new time to delivery estimate by using the new cost driver. Scale factors for this estimate are considered the same as the previous estimation.

Table 12 Feature estimate

Feature A	7.5 months
Feature B	5.5 months
Feature C	5 months
Feature D	2.5 months

We can observe that the new results show a slightly better estimation compare to the original ones. The biggest concern in this estimation relates to Feature C. This specific feature is blocked due to external dependencies at module levels, and without including the timeline for delivery of the external project, we cannot provide an accurate delivery time. If this type of impediments exists in a project, the feature is not ready to work anymore and cannot be prioritized although, by including the environment readiness the WSJF score of this feature might be lower than the other features in the backlog and potentially can be postponed for the next PI. Now, we can calculate the score with our new data:

Table 13 Feature WSFJ with Environment readiness

	Feature A	Feature B	Feature C	Feature D
WSJF	0.6	0.9	0.5	0.6

Table [13] shows that the WSFJ score of Feature A and B reduced after the right estimation of their size is provided. If we included other features in the backlog and calculating WSFJ with this new method, none of this feature might get a high priority for the increment.

5. Other factors and future study

COCOMO-II provides scheduling by considering some historical data also determines possible required personnel for delivering the requirements. In this study, we used the time to delivery instead of personnel month as the structure of scrum teams are not the same as traditional projects, converting PM to schedule without using COCOMO-II could eliminate the accuracy of our results and create an entirely new approach but some of the values that are used for the time to delivery should be adjusted for Scrum methodology. The other factor is that COCOMO-II is not calibrated for small projects, so the result for Feature C and Feature D might need to be calibrated. Besides, Scrum has several ceremonies that consume time from execution teams. In traditional project management, processes such as risk management, planning, and scope change meetings are run and governed by the project manager, but Scrum team members are part of this process, so we need to add overhead to delivery time. In many companies, they consider 20% of their team capacity dedicated to team management. This overhead will not have an impact on features priorities but can explain some differences between the estimates and actual results.

This study does not consider requirement change. One of the primary reasons that companies are eager to use Agile methods is its responsiveness to project changes. This is a more complicated issue as SAFe 3 months in advance planning does not provide an Agile response to changes. The other factor that could also impact the team performance in SAFe is the high-level design provided by the program team. In modern software, developing the code is the architect, and many IT companies try to avoid flow chart design for their software. They believe methods like Test-Driven Development (TDD) can provide a better HLD, so flowchart should be avoided unless they are used at the enterprise level that can provide the technical vision for the company. When the program team assigns a feature that already has HLD, they force the execution team to follow a specific rule that they might not agree with it. Scrum teams are the closest teams to the code, and no one understands an application better than developers who are going to implement it. Therefore, KSLOC cannot be estimated correctly as the program team measures it.

Another challenge was the silos of knowledge that had built up over time, not just at the BO level but throughout all of Product Management [6] so even though at a very high level it seems teams are organized under one train that shared the same goals and objectives, work cannot be easily transferred from one Scrum backlog to another. There is a misconception that some updates and knowledge sharing meetings, run by delivery managers, and product owners can overcome this issue, but, sometimes the transition process takes a month to be complete while this overhead easily gets ignored. Agile maturity has a direct impact on this issue. An agile maturity assessment is a way to evaluate how a team is improving their ability to be agile over time. [9]. Before and during SAFe adoption, organizations require a uniform model for assessing the current state and progress, and for establishing a roadmap for the initiative. [10]. In [10] maturity model is provided that can show the path to the companies to adopt Agile and SAFe practices. In COCOMO II some scale factors like TEAM and personnel cost driver category can be used to calculate this impact. However, in Agile, it is not sufficient to only consider them. In [10], they found teams accomplished Agile maturity via a dynamic evolution based on the pursuit of specific outcomes. Among their finding, practices learning, team conduct, and pace of deliveries have a significant rule in SAFe as Agile is built based on trust and team professionalism

without having constant process improvement teams cannot succeed.

For the future study, it is required to consider the mentioned factors and log the historical data to adjust COCOMO-II scale factors and cost drivers for SAFe. One possible approach is to compare Work Breakdown Structure (WBS) and bottom-up approach to the presented solutoin and try to discover the root cause of the discrepancy between these estimates. Also, applying computer science from different fields like AI or use other mathematical modeling approaches can enhance the accuracy of our sizing method. As an example, fuzzy logic can calculate the effort estimation in Agile development. Fuzzy logic is an effective strategy to tackle the practical issues which incorporate the effort less ness and adaptability [8]. It handles the issues with loose and inadequate information, which depends on the fuzzy set hypothesis based on limited information.[8] The benefit of using range is that the software developers can easily predict the effort of a project that has a different model [8].

6. Conclusions

This study demonstrates there is a demand to measure the feature size in SAFe, and it has a direct impact on the priority of work in the backlog. T-shirt sizing used in this study could not produce a proper evaluation for the project size. Train release engineers and project managers in SAFe need to acknowledge that in many areas, they need to recognize different perspectives in their process. Although COCOMO-II and upcoming COCOMO-III do not include Agile in their proposals, we showed that Scaled Agile could take advantage of a COCOMO-II estimation.

We demonstrated an additional cost factor benefits the job size estimation, and the results are so closer to reality compare to what the program team provided it; nonetheless, it could not eliminate the inconsistency. If this method continues to be used for at least a year, a more substantial outcome can be achieved.

7. References

[1] Agilenutshell.com. 2020. What Is Agile? [online] Available at: <http://www.agilenutshell.com/> [Accessed 9 March 2020].

[2] Scaled Agile. 2020. What Is Safe | Scaled Agile. [online] Available at: <https://www.scaledagile.com/enterprise-solutions/what-is-safe/> [Accessed 9 March 2020].

[3] Rajput, G. and Litoriya, R., 2014. Corad Agile Method for Agile Software Cost Estimation. OALib, 01(03), pp.1-13.

[4] Agileestimator.com. 2020. Use The COCOMO Suite To Estimate Staffing And Schedule For Agile Projects. [online] Available at: <http://www.agileestimator.com/2019/05/12/use-the-cocomo-suite-to-estimate-staffing-and-schedule-for-agile-projects/> [Accessed 9 March 2020].

[5] Silva, Caio Cestari, and Alfredo Goldman. “Agile Methods Adoption on Software Development: A Pilot Review.” 2014 Agile Conference, 2014, doi:10.1109/agile.2014.14.

[6] Kalliney, Marie. “Transitioning from Agile Development to Enterprise Product Management Agility.” 2009 Agile Conference, 2009, doi:10.1109/agile.2009.64.

[7] Hanschke, Sebastian, et al. “Integrating Agile Software Development and Enterprise Architecture Management.” 2015 48th Hawaii International Conference on System Sciences, 2015, doi:10.1109/hicss.2015.492.

[8] Saini, Abhishek, et al. “Effort Estimation of Agile Development Using Fuzzy Logic.” 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2018, doi:10.1109/icrito.2018.8748381.

[9] Balbes12/15/2015, Mark. “How Agile Are You? Let's Actually Measure It! (Part 0: Introduction).” ADTmag, adtmag.com/articles/2015/12/15/balbes-agile-model-0-intro.aspx.

[10] Stojanov, Igor, et al. “A Maturity Model for Scaling Agile Development.” 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, 2015, doi:10.1109/seaa.2015.29.

[11] Fontana, Rafaela Mantovani, et al. “Agile Compass: A Tool for Identifying Maturity in Agile Software-Development Teams.” IEEE Software, vol. 32, no. 6, 2015, pp. 20–23., doi:10.1109/ms.2015.135.

[12] Rautiainen, K, et al. “Supporting Scaling Agile with Portfolio Management: Case Paf.com.” 2011 44th Hawaii International Conference on System Sciences, 2011, doi:10.1109/hicss.2011.390.

[13] Kang, Sungjoo, et al. “Model-Based Dynamic Cost Estimation and Tracking Method for Agile Software Development.” 2010 IEEE/ACIS 9th International Conference on Computer and Information Science, 2010, doi:10.1109/icis.2010.126.

[14] Agarwal, Aman. “Software Engineering: COCOMO II Model.” GeeksforGeeks, 11 Mar. 2019, www.geeksforgeeks.org/software-engineering-cocomo-ii-model/.

[15] Aggarwal, K. K. Software Engineering. New Age Int'L Publishers, 2005. PP 165

[16] “What Is Scrum Methodology?” What Is Scrum Methodology & Scrum Project Management, 16 Feb. 2020, resources.collab.net/agile-101/what-is-scrum.