AgileCritPath: Identifying Critical Tasks in Agile Environments

Rachel Vital
PESC/COPPE
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
rachelvital@cos.ufrj.br

Glaucia Melo
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
gmelo@uwaterloo.ca

Toacy Oliveira
PESC/COPPE
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
toacy@cos.ufrj.br

Paulo Alencar
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
palencar@uwaterloo.ca

Don Cowan
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
dcowan@uwaterloo.ca

***Abstract*—Planning and monitoring the execution of software development activities in agile environments are not trivial procedures. One of the main flaws of agile planning is not considering the dependencies that exist between project tasks. Dependencies between tasks found in software development project plans may lead to the emergence of critical paths, where tasks need to be handled in a strict sequence because the completion of some tasks depends on the completion of others. Not managing such critical paths may reduce team performance and delay product delivery. We performed a study to demonstrate that not identifying dependencies may impair team performance and even increase the risk of costly project delays. The study is divided into three parts: (1) an exploratory study performed in the industry; (2) the development of a tool, AgileCritPath, as a way to support development teams in identifying critical project tasks; and (3) an *in vivo* evaluation of AgileCritPath based on the Technology Acceptance Model (TAM). The results of the exploratory study provided empirical evidence that there is a need to identify and control dependencies between the tasks in the development of software in agile environments. Using the AgileCritPath tool, allowed us to introduce the Critical Path Method concepts in an agile software development organization. Moreover, the *in vivo* evaluation demonstrated the benefits of managing tasks dependencies.**

***Index Terms*—Software Engineering, Critical Path Method, Agile, Software Development.**

I. Introduction

The software development principles behind the "Manifesto for Agile Software Development" are widely used in software development project management. Although the adoption of such agile principles brings numerous benefits, including the delivery of software products on time and budget [1], estimating and planning projects using agile methodologies is still a complex process. Part of this complexity comes from the difficulty of identifying dependencies between tasks, which is a critical factor for coping with agile project failures [2].

Dependencies between tasks often lead to decreases in team's agility level [3] and can adversely impact the delivery time of software products. For Bick and his colleagues [4], managing task dependencies is a fundamental issue in agile software development, because it can be used as a basis to define the coordination between project tasks. Additionally, proper identification of task dependencies is important to maximize project efficiency and reduce risks [5], [6], [7], [8], [9], [10].

DOI reference number: 10.18293/SEKE2019-004

The critical need to manage dependencies found in agile-based software development projects is a topic that has been already explored in the literature [5], [11], [4], [12], but, currently, empirical studies on the subject are still scarce. There is a clear need to investigate this topic further, given that the lack of knowledge about task dependencies may have consequences for project duration, coordination, risk, and efficiency.

To understand the problem of identifying the dependencies properly between agile projects and software development tasks, we conducted an exploratory study in a software development organization. Through this study, we obtained empirical evidence that this is a real problem that can negatively affect software development organizations, especially when they use agile methods. After performing the exploratory study, we used the techniques of the Critical Path Method (CPM) to inform a team in a specific software development organization that already uses agile methods about the critical tasks. We have also developed a tool, which we call AgileCritPath, to evaluate the adoption of CPM in an organization. The evaluation followed the Technology Acceptance Model (TAM) [13] approach.

This paper is structured as follows. After a brief introduction in Section I, Section II describes supporting concepts. Section III presents an Exploratory Study. Section IV presents the AgileCritPath tool and a TAM evaluation of this tool. Finally, Section VI presents conclusions and future work.

II. Background

The increased adoption of agile practices has caused traditional project management to be redefined.

Many agile methods provide some level of management for tasks, but out of the main techniques applied [14] none guarantees that teams follow what was planned. Cohn [2], in his book "Agile Estimating and Planning", addresses the main flaws of agile planning. Among them, the author cites that tasks are not independent and that it is an error for the agile

teams to plan the tasks as if they were separate, with no need for task coordination.

In agile methods, the technical work of the development team is defined through tasks. Each team estimates tasks and, generally, they represent a small part of the expected work [2]. Tasks can be visualized using the Kanban board presented in Figure 1. Kanban boards visually depict work at various stages of a process using cards to represent tasks and columns to represent each stage of the process. Cards are moved from left to right to show progress and to help coordinate the teams performing the work. The Kanban board is a visual approach that teams use to monitor task execution.

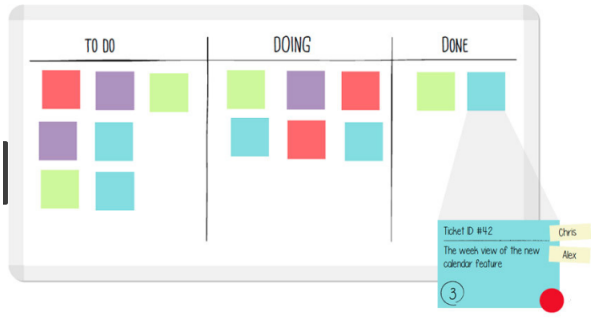

Fig. 1. Kanban board example.

Progress monitoring is an essential activity in Software management, whereby it ensures that a project plan advances according to budget, schedule, and quality expectations [15], [16]. The implementation of progress monitoring mechanisms in an agile environment is fundamental to the success of the project.

Although the Kanban framework is widely adopted in organizations, it does not show task dependencies. It is not possible to see on the Kanban board which tasks can block the execution of others and, for this reason, the flow of task execution becomes unclear to the team.

The Agile Kanban method lacks a mechanism for progress tracking. Thus, it needs to be integrated with other methods because it does not have a standard definition for software development and its specific practices have not yet been rigorously defined [12].

However, although managers can have, based on the Kanban board, support to understand progress status and progress better, they can not get information about the impact of the execution of the flow of tasks, especially when it comes to task dependencies.

III. Exploratory Study - Identifying Critical Tasks

To correctly understand the dynamics of the organization concerning project planning in an agile environment, we conducted an exploratory study in a specific software development organization. Exploratory studies have proved to be adequate in software engineering to study new ideas [17].

Through the exploratory study, we have: (1) observed the existence of dependencies between tasks in real software development projects that use agile methods; (2) applied concepts based on the Critical Path Method to identify critical tasks in projects that use agile methods; (3) collected, analyzed and discussed the results. The exploratory study is divided into the following steps: definition of the study, data collection, data analysis, and presentation of results.

A. Definition

The study was conducted in a Brazilian software development company that has a staff of 30 employees, most of whom are software developers. The organization has been using agile methods for at least 10 years. Teams are small, usually groups of 3 to 8 people.

B. Data collection

The task execution log has been extracted from the Redmine[1] project management tool. Through the organization's task management system, we were able to identify which tasks were planned and which tasks were performed. We retrieved data from three iterations of the same project for analysis.

C. Data analysis

In this step, we identified the dependencies between tasks. The dependency identification was performed with the help of an experienced developer who was familiar with the scope of the tasks, the processes of the organization, and knew the technological architecture used in the project. Task dependencies were classified into Process and Context dependencies, following a classification suggested in a taxonomy of dependencies [5].

D. Results and discussion

From the analysis of the data, the projection of the maximum effort rate per period was obtained. Therefore, regardless of the number of people on the team, we can not plan activities with effort higher than the maximum effort rate. For the calculation of the maximum effort rate, we consider the estimated effort of the tasks performed by the team and their respective dependencies.

Figure 2 illustrates the planning performed by the team (blue line) and the maximum effort rate of the tasks (red line). The red line represents the shortest time to complete the activities and indicates the maximum rate of production (speed) at which the team can work. Regardless of the number of people, the tasks will not be completed before this deadline (red line). In this case, we can see that the team planned a delivery that can not be performed on the expected date.

Table I presents the number of dependencies identified in the study and the number of tasks defined by the team, which are extracted through the task execution log. The number of dependencies is higher than the number of tasks, within the three analyzed iterations.

[1] redmine.org

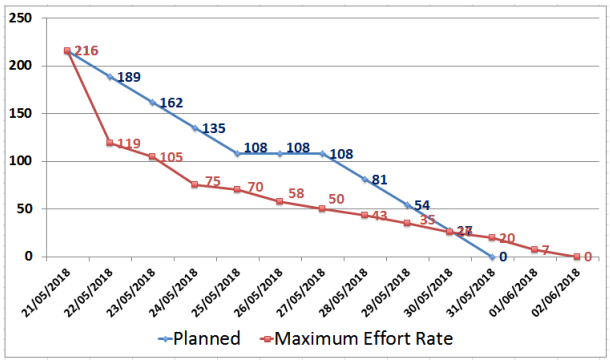

Fig. 2. The burndown graph with maximum effort rate.

TABLE I
PERCEIVED DEPENDENCIES.

Release	Task	Dep	% Dep	Process Dep	Business Dep
Release 2.12	100	96	96%	52	44
Release 2.13	69	77	112%	53	24
Release 2.14	70	84	120%	67	17

Our results show that handling task dependencies may not be straightforward when there are many tasks. Another factor we noticed was that there were several tasks, and consequently, dependencies, that are were not planned but were identified during the iteration execution.

During the analysis of the Iteration execution log, we verified that some tasks were discovered during the execution of the iteration and, consequently, the execution flow of the tasks changed because the new tasks had dependencies that should have been considered. This shows that the identification of critical tasks should occur throughout an iteration, and not only at the end.

Based on the exploratory study we conclude that there is evidence that when considering task dependencies it is possible to find the critical tasks, which if delayed, can have serious consequences, including the product release delays. Because of this evidence, we decided to develop the work further and implement AgileCritPath, a tool that could help developers identify critical tasks. The tool implementation and details are presented in Section IV.

IV. The AgileCritPath Tool

The AgileCritPath tool implements the Critical Path Method to be used in agile environments and support teams to identify and prioritize critical tasks that can block the execution of other tasks and cause costly project delays.

A. Identifying Critical Tasks

During the exploratory study, we identified that some tasks compromise or block the flow of execution of other tasks. These tasks are referred to as Critical Tasks because they can decrease the agility level of the team.

The critical path is the longest duration path through the network. The tasks that lie on the critical path cannot be delayed without delaying the release. In agile environments, Critical Tasks are identified by building a network diagram of tasks. This network diagram is built to represent task dependencies. Figure 3 presents an example of a network diagram with seven tasks. The solid lines represent dependencies between tasks, and the dashed lines represent the association of the tasks from the beginning to the end of the graph. Critical tasks are identified as the tasks that make up the longest delivery path.

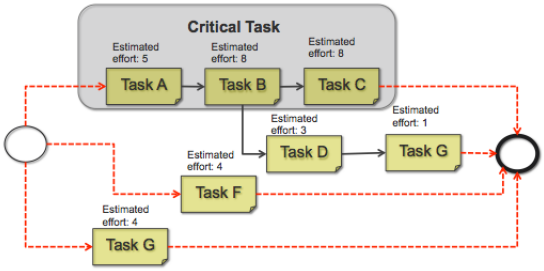

Fig. 3. Network Diagram with Critical Tasks.

B. Proposed Model

Figure 4 illustrates in detail the mechanism used to find out the Critical Task in GitHub projects.

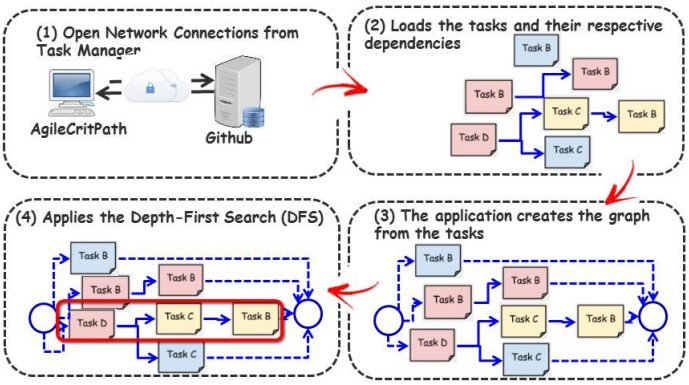

Fig. 4. Illustration of AgileCritPath.

In step (1), the application connects to the task manager to load all tasks and their dependencies. In step (2), the associations between tasks are created according to the dependencies. In step (3), the resulting graph is created. Tasks without predecessors are automatically linked to the initial node of the graph. The tasks without successors are automatically linked to the final node of the graph. In step (4), the Depth-First Search (DFS) algorithm is applied to find all paths in the graph. While reading the path, the total effort to complete the task flow is calculated.

All the found paths are displayed in descending order, from the longest path to the shortest path, according to the calculated effort for each path. Next to the path listing is the status of the task, the developer responsible for the execution of the task, and the effort of each task that makes up the path. The path size is calculated by the sum of the planned effort

reported in the individual tasks that are part of the path. If the task is finished, we consider the effort to complete the task.

Knowing all the paths in the network task is important because we can identify the "Near-Critical Paths". Other important paths through are considered the "Near-Critical Paths" if they are at risk of becoming the Critical Path. To make this information visible to the team, we display all the paths found in the task network.

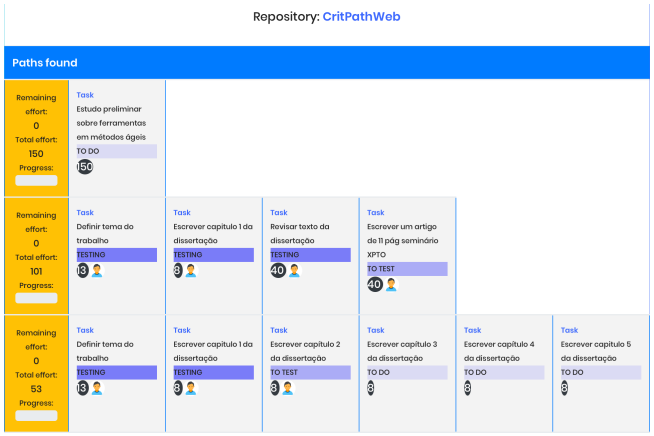

Fig. 5. Query results.

Figure 5 presents the result of querying the paths of a task network from a GitHub repository.

C. Architecture model

AgileCritPath was developed following the architecture presented in Figure 6.

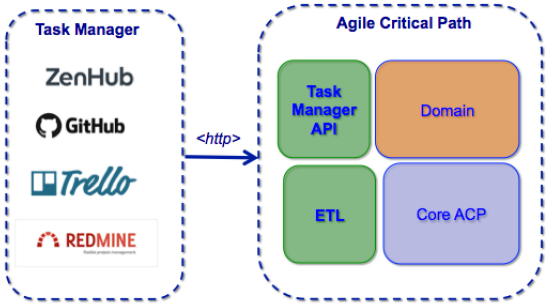

Fig. 6. The AgileCritPath architecture.

The application works autonomously and connects to the task manager through an integration API (task manager API). The Extract, Transform and Load (ETL) module performs the necessary information processing, according to the information available in the task manager. An ETL module was developed to meet the integration requirements with GitHub, ZenHub and Redmine. The ETL loads objects from the application domain, which works in isolation from the task management tools. The Core ACP module loads a task-oriented graph and implements a Depth-First Search algorithm that searches for all paths in the task network.

Planning in an agile environment is different from approaches used in traditional plan-oriented software development models [18]. Rather than employing plans in projects based on a set of predefined factors and constraints, agile models rely on the human factor to self-organize. In this work, we seek to offer systemic support, allowing the information to be accessible to all team members, and the decisions about the order of task execution to be made at any time.

V. The AgileCritPath Tool Evaluation

The evaluation of the AgileCritPath tool was performed according to the Technology Acceptance Model (TAM). The TAM approach was proposed by Davis in 1989 [19] and suggests the acceptance of a new IT technology depending on two variables: (1) perceived ease of use and (2) perceived utility usefulness. For Davis, people tend to use or not use technology to improve their performance at work - perceived utility. However, even if a person understands that a particular technology is useful, its use may be impaired if the application is too complicated, so effort does not compensate for use - perceived ease.

A. Procedures

The general objective of using TAM was to evaluate the potential of the AgileCritPath tool in an industrial environment. The evaluation was performed in a company that has been working in the area of software engineering for 20 years and adopts development processes based on agile practices. The project used in the evaluation was selected by the organization. The selected company uses Redmine as a task management tool. For the execution of the study, we modified our tool so it could access Redmine.

B. Execution

The study was performed in-vivo. *In-vivo* studies are studies that involve people in their own working environment under realistic conditions [20]. Case studies made in an industrial environment are an important type of *in-vivo* study since they allow the analysis of a particular process in the context of a software life cycle [21] .

To ensure that the study did not impact the organization's software development process, we restricted ourselves to observing the tasks performed in the team's daily routine and added to their lists the activities required to use the tool, as presented in Figure 7.

Figure 7 presents the activities performed in the organization. The activities in red are the activities we included so that CPM could be used by the team.

At the iteration planning meeting, the team began to include task dependencies. During the execution of the iteration, the team performs daily meetings (daily Scrum meeting). As suggested in Scrum [22], daily meetings should last 15 minutes, where the team discusses what has been done in the last 24 hours, the plan for the next 24 hours, and what task impediments (anything that keeps a team from being productive) occurred. The meeting is held in front of a

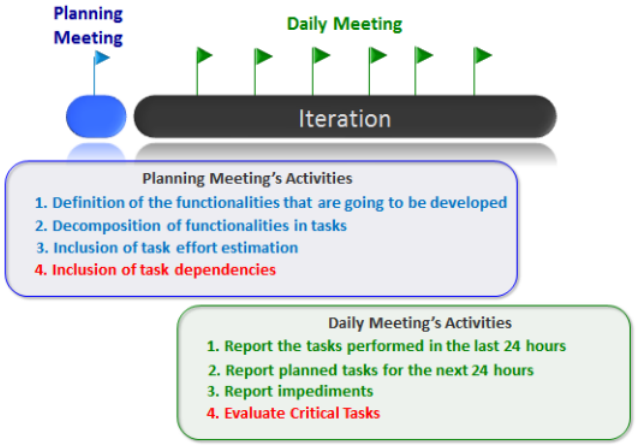

Fig. 7. The activities performed during an iteration.

Kanban board. The visualization of the paths calculated by the AgileCritPath tool was presented at the end of the daily meetings.

During the planning and execution of the iteration, the participants had access to the AgileCritPath tool, and in the daily meetings, the tasks that were part of the critical path were presented to the team. The iteration lasted two weeks, and at the end, the participants were asked to answer the questions directed towards the evaluation of the tool.

At the end of the iteration, the participants completed the Participation Consent and Clearance Form, Participants Characterization Form and the TAM Model Evaluation. The forms were completed individually and without any contact between participants.

C. Discussion

A team of six employees participated in the study. Participation was free and voluntary. The questionnaire was sent to all developers of the company. The participants, in general, have a bachelor's degree, except for one participant, who is in the process of concluding a bachelor's degree. The academic experience varies greatly among the research group since the team has 2 senior, 1 junior and 1 trainee members. All participants are familiar with the basics of project management and are knowledgeable about the Scrum methodology.

From the results of the study, based on the Perceived Utility evaluation criterion, we verified that the participants agreed that the use of the tool could improve the productivity and quality of the work of the team members. All participants recognized the tool as useful for performing their tasks. Regarding the Perceived Usability Facility variable, the team indicated that identifying dependencies between tasks and identifying the Critical Path as reasonably difficult activities. Even though the team has indicated difficulties in identifying dependencies and evaluating the critical path, 75% of respondents felt inclined to use the tool during the planning and execution of tasks. Despite having to make a more significant effort in task dependency identification, the team recognized the benefits that the use of the tool can provide.

The participants were asked to identify the benefits and limitations of the tool. The key highlighted benefits were: the visualization of dependencies between tasks, as a facilitator to determine which tasks should be prioritized, and the improved visualization of the total planned effort to complete critical path tasks. As a tool limitation, the participants recognized that it would be nice if the tool could display estimated versus realized information of the tasks in progress and could provide more metrics for monitoring work progress.

During the daily meetings using the tool, we were able to capture observations made by the team as follows:

- The team observed that using the tool allows everyone on the team to get a sense of what tasks are critical;
- The team found it interesting to leave the critical tasks to the most experienced developers on the project, especially when the deadlines for these tasks are tight;
- Visualizing the critical path is also a way of evaluating the most complex points of the project;
- The team observed that the task priority they provide at the time of task planning did not make sense and, in some cases, the team acknowledges that they might have had a greater gain prioritizing critical path tasks;
- The team evaluated the critical path analysis as a tool complementary to the Kanban framework because in the Kanban framework they could not keep track of the dependencies between the tasks.

The lack of knowledge about dependencies between planned tasks in agile environments emerges in a misaligned business plan that the team may not be able to execute. Also, the lack of awareness about the dependencies that exist between the tasks of the software development process constitutes a possible explanation for inefficient team coordination and project delays.

The AgileCritPath tool enables organizations to use dependency information across tasks to improve task prioritization in agile environments by identifying which dependencies can compromise or block the flow of task execution.

In this study, the evaluation of the usability of the AgileCritPath tool in the industry was conducted. Usability is one of the aspects related to the quality of use of systems, being one of the most important acceptance criteria for interactive applications in general, and in particular for Web applications [23]. The acceptance of technology is related to the quality and use of its systems, the quality of the provided information and user satisfaction [24]. Through the use of the acceptance model, it was possible to evaluate the use of the AgileCritPath tool in an industrial environment.

VI. Conclusion

Agile approaches are based on the idea that developers can self-organize and perform their work collaboratively [4]. However, some studies suggest that large and complex software development projects can benefit from the combination of the flexibility inherent in agile teamwork and models that support a plan-oriented structure [25], [26], [27].

Bick et al. [4] and Badampudi et al. [18] propose that organizations should continue to adopt agile methods, including the established practices of traditional methodologies that guarantee more predictability, reliability, stability and effective use of resources. In addition, through analysis of dependencies, the team can evaluate the shorter time for product delivery. Not considering the dependencies between tasks can generate delivery plans that are not aligned with the reality of the organization.

Considering dependencies on software projects is crucial to identifying critical tasks. Critical tasks are tasks that can delay the execution of others and thereby delay the delivery of the product. In this paper, we present an exploratory study that demonstrated that in a real software development scenario, the number of dependencies could be large and difficult to manage. Through the results of the exploratory study, we proposed the use of the Critical Path Method (CPM) concepts to identify critical tasks in agile projects. Critical Path Method has already consolidated in traditional project management models. In Agile Environments, this technique can help the team identify critical tasks, and thus direct their efforts toward tasks that can impact the development of other tasks.

The proposal presented in this article was materialized in the AgileCritPath tool, which allows development teams to have a view of dependencies between the tasks and, therefore, identify at any time which tasks are critical. The tool is compatible with any agile methodology and can support development teams to make decisions easily and quickly. The tool was developed in open source format and is available on GitHub (https://github.com/RachelVital/CritPath).

The evaluation of the use of the tool was based on the TAM technological acceptance model. During the evaluation, we were also able to gather insights and encourage the adoption of the tool in the industry. Through the responses obtained in the TAM application, we were able to evaluate that, despite the additional effort to identify the dependencies, the use of the CPM in agile environments proved to be relatively simple and viable in the selected organization. At the end of the evaluation of the use of technology, we could verify that the whole team was able to perceive the utility of the identification of critical tasks.

As future work, we plan to evaluate the use of AgileCritPath in continuous software development and DevOps environments. The application of the acceptance model could also be conducted in other software development companies.

ACKNOWLEDGMENT

The authors thank the Natural Sciences and Engineering Research Council of Canada (NSERC), the Emerging Leaders in the Americas Program (ELAP) and MITACS.

REFERENCES

[1] J. Lynch. Standish group 2015 chaos report - q&a with jennifer lynch. [Online]. Available: https://www.infoq.com/articles/standish-chaos-2015

[2] M. Cohn, *Agile Estimating and Planning*. Pearson Education, google-Books-ID: BuFWHffRJssC.

[3] C. D. W. Lomas, J. Wilkinson, P. G. Maropoulos, and P. C. Matthews, "Measuring design process agility for the single company product development process." vol. 9, no. 2, pp. 105–112.

[4] S. Bick, K. Spohrer, R. Hoda, A. Scheerer, and A. Heinzl, "Coordination challenges in large-scale software development: A case study of planning misalignment in hybrid settings," vol. 44, no. 10, pp. 932–950.

[5] D. E. Strode, "A dependency taxonomy for agile software development projects," vol. 18, no. 1, pp. 23–46. [Online]. Available: http://dx.doi.org/10.1007/s10796-015-9574-1

[6] M. Korkala and F. Maurer, "Waste identification as the means for improving communication in globally distributed agile software development," vol. 95, pp. 122–140.

[7] M. Shen, G.-H. Tzeng, and D.-R. Liu, "Multi-criteria task assignment in workflow management systems," in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*. IEEE, pp. 9–pp.

[8] J. Duggan, J. Byrne, and G. J. Lyons, "A task allocation optimizer for software construction," vol. 21, no. 3, pp. 76–82.

[9] Y. Jiang and J. Jiang, "Contextual resource negotiation-based task allocation and load balancing in complex software systems," no. 5, pp. 641–653.

[10] J. Sutherland and K. Schwaber, "The scrum guide. the definitive guide to scrum: The rules of the game."

[11] W. Aslam and F. Ijaz, "A quantitative framework for task allocation in distributed agile software development," vol. 6, pp. 15 380–15 390.

[12] H. Alaidaros, M. Omar, and R. Romli, "Identification of criteria affecting software project monitoring task of agile kanban method," in *AIP Conference Proceedings*, vol. 2016. AIP Publishing, p. 020021.

[13] Y. Lee, K. A. Kozar, and K. R. Larsen, "The technology acceptance model: Past, present, and future," vol. 12, no. 1, p. 50.

[14] VersionOne. 12th annual state of agile report. [Online]. Available: https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report

[15] M. L. Despa, "Comparative study on software development methodologies," vol. 5, no. 3, pp. 37–56.

[16] Hazır, "A review of analytical models, approaches and decision support tools in project monitoring and control," vol. 33, no. 4, pp. 808–815.

[17] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," vol. 14, no. 2, pp. 131–164. [Online]. Available: http://link-springer-com.ez29.capes.proxy.ufrj.br/article/10.1007/s10664-008-9102-8

[18] D. Badampudi, S. A. Fricker, and A. M. Moreno, "Perspectives on productivity and delays in large-scale agile projects," in *International Conference on Agile Software Development*. Springer, pp. 180–194.

[19] F. D. Davis, "A technology acceptance model for empirically testing new end-user information systems : theory and results," http://hdl.handle.net/1721.1/15192, 7 1986, thesis (Ph. D.)–Massachusetts Institute of Technology, Sloan School of Management, 1986.; MICROFICHE COPY AVAILABLE IN ARCHIVES AND DEWEY.; Bibliography: leaves 233-250.

[20] G. H. Travassos and M. O. Barros, "Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering," in *2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering*, pp. 117–130.

[21] F. Shull, J. Carver, and G. H. Travassos, "An empirical methodology for introducing software processes," in *ACM SIGSOFT Software Engineering Notes*, vol. 26. ACM, pp. 288–296.

[22] P. Deemer, G. Benefield, C. Larman, and B. Vodde. The scrum primer version 2.0.

[23] E. Insfran and A. Fernandez, "A systematic review of usability evaluation in web development," in *International Conference on Web Information Systems Engineering*. Springer, pp. 81–91.

[24] S. Petter, W. DeLone, and E. R. McLean, "The past, present, and future of" IS success"," vol. 13, no. 5, p. 341.

[25] J. B. Barlow, J. S. Giboney, M. J. Keith, D. W. Wilson, and R. M. Schuetzler. Overview and guidance on agile development in large organizations.

[26] L. Cao, K. Mohan, P. Xu, and B. Ramesh, "How extreme does extreme programming have to be? adapting XP practices to large-scale projects," in *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*. IEEE, pp. 10–pp.

[27] N. Ramasubbu, A. Bharadwaj, and G. K. Tayi, "Software process diversity: conceptualization, measurement, and analysis of impact on project performance."