

**A DECISION SUPPORT SYSTEM FOR SPRINT PLANNING
IN SCRUM PRACTICE**

Alhejab Shawqi Alhazmi

A Thesis Submitted to the Faculty of
The College of Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of
Master of Science

Florida Atlantic University

Boca Raton, FL

May 2018

Copyright 2018 by Alhejab Shawqi Alhazmi

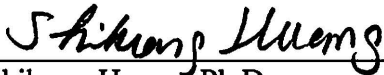
**A DECISION SUPPORT SYSTEM FOR SPRINT PLANNING
IN SCRUM PRACTICE**

by


Alhejab Shawqi Alhazmi

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Shihong Huang, Department of Computer & Electrical Engineering and Computer Science, and has been approved by all members of the supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

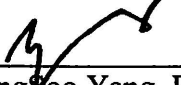
SUPERVISORY COMMITTEE:



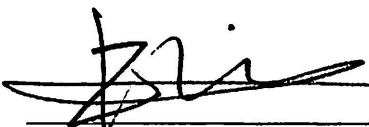
Shihong Huang, Ph.D.
Thesis Advisor



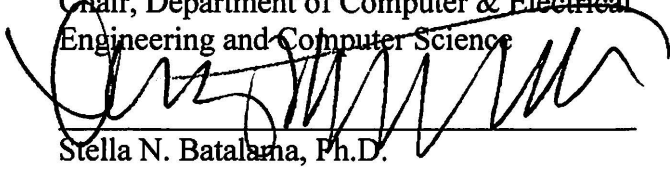
Ionut Cardei, Ph.D.



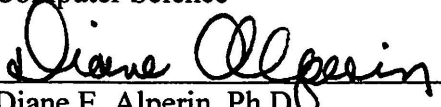
Kwangsoo Yang, Ph.D.



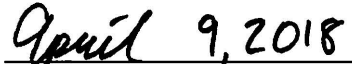
Nurgun Erdol, Ph.D.
Chair, Department of Computer & Electrical
Engineering and Computer Science



Stella N. Batalama, Ph.D.
Dean, College of Engineering and
Computer Science



Diane E. Alperin, Ph.D.
Interim Dean, Graduate College



Date

ACKNOWLEDGEMENTS

I would like to express sincere gratitude to my advisor Dr. Shihong Huang for her persistence, patience, friendly advice, and encouragement during my journey to get this degree and thanks to the committee members for their guidance and support through the process of working on this thesis. I would also like to thank my family for being encouraging and supporting me during my academic career. I wish also to thank all the Software Engineering Lab members at FAU for their support. Finally, thank you to all my friends for always encouraging me to achieve this academic career.

ABSTRACT

Author: Alhejab Shawqi Alhazmi
Title: A Decision Support System for Sprint Planning in Scrum Practice
Institution: Florida Atlantic University
Thesis Advisor: Dr. Shihong Huang
Degree: Master of Science
Year: 2018

Scrum is one of the Agile software development processes broadly adopted in industry. Scrum promotes frequent customer involvements and incremental short release. Sprint planning is a critical step in Scrum that sets up next release goals and lays out plans to achieve those goals. This thesis presents a Sprint Planning dEcision Support System (SPESS) which is a tool to assist the managers for Sprint planning. Among considering other Sprint planning factors, SPESS takes into consideration developer competency, developer seniority and task dependency. The results are that the assignments of the tasks of each Sprint to developers guarantee that each team member contributes to their fullest potential, and project planning is optimized for the shortest possible time.

Keywords—Scrum, Sprint planning, planning poker, competence, task dependence, Hungarian algorithm, Essence.

DEDICATION

This work is dedicated to my parents, who supported my goals along my life, to my wife Mona and my kids Abdulaziz and Abdulmalik, who encouraged and supported me, and to my brothers and sisters who made all the difference in my life.

**DECISION SUPPORT SYSTEM FOR SPRINT PLANNING
IN SCRUM PRACTICE**

LIST OF TABLES	x
LIST OF FIGURES	xiii
LIST OF ABBREVIATIONS.....	xiv
CHAPTER 1: INTRODUCTION	1
1.1 Scrum Overview	1
1.1.1 Scrum Roles	2
1.1.2 Scrum Artifacts	3
1.2 Project Planning	4
1.3 Problem Statement	4
1.4 Work Motivation and Objective	4
1.5 Contributions.....	5
1.6 Related Work	5
1.7 Thesis Organization	6
CHAPTER 2: THE RESEARCH METHODOLOGY	8
2.1 Research Methodology	8
2.2 Improved Scrum.....	9
2.3 Attributes of the Research Methodology	10
2.3.1 Planning Poker	10

2.3.2 The Developer's Seniority	11
2.3.3 The Developer's Competence	12
2.3.4 The Task Dependency.....	13
2.3.5 The Hungarian Algorithm.....	14
CHAPTER 3: SPESS FRAMEWORK.....	16
3.1 Product Owner Inputs	17
3.2 Developers Inputs	18
3.3 Middle Layer.....	18
3.4 Task Assignment Model	19
3.4.1 Process – 1	19
3.4.2 The Hungarian Algorithm.....	19
3.4.3 Process – 2	20
3.5 Output	20
CHAPTER 4: ILLUSTRATION OF USING SPESS TOOL	21
4.1 Collecting Inputs.....	21
4.2 Preparing for the Tables.....	24
4.3 Process - 1	26
4.4 Applying Hungarian Algorithm.....	27
4.5 Process - 2	28
4.6 Sprint's Iterations.....	30
4.6.1 Iteration 1	31
4.6.2 Iteration 2	31
4.6.3 Iteration 3	33

4.6.4 Iteration 4	34
4.6.5 Iteration 5	36
4.6.6 Iteration 6	37
4.6.7 Iteration 7	38
4.7 Outputs	40
4.7.1 The Result of Suggested Sprint.....	40
4.7.2 The Result of Real Case Study	42
CHAPTER 5: DISCUSSION AND CONCLUSION	43
5.1 Discussion, Recommendations and Limitations	43
5.1.1 Analysis of Results and Discussion	43
5.1.2 Recommendations for Using SPSS	44
5.1.3 SPSS Limitations	46
5.2 Conclusion and Future Work	49
5.2.1 Conclusion	49
5.2.2 Future Work	49
REFERENCES	51

LIST OF TABLES

Table 2-1: Competence Levels and Their Checklist [22]	13
Table 4-1: Developers' Seniority	22
Table 4-2: Sprint Competence	22
Table 4-3: Main-Table	22
Table 4-4: Dependency-Table.....	23
Table 4-5: Main-Table (ordered)	24
Table 4-6: Time-Table (initial)	24
Table 4-7: Competence-Table.....	26
Table 4-8: Time-Table	26
Table 4-9: Free-Tasks-Table (Iteration 1).....	27
Table 4-10: To-Do-Table (Iteration 1).....	31
Table 4-11: Iterations-Table (Iteration 1)	31
Table 4-12: Dependency-Table (Iteration 2)	31
Table 4-13: Time-Table (Iteration 2)	32
Table 4-14: Free-Tasks-Table (Iteration 2).....	32
Table 4-15: To-Do-Table (Iteration 2).....	33
Table 4-16: Iterations-Table (Iteration 2)	33
Table 4-17: Dependency-Table (Iteration 3)	33
Table 4-18: Time-Table (Iteration 3)	33
Table 4-19: Free-Tasks-Table (Iteration 3).....	34

Table 4-20: To-Do-Table (Iteration 3).....	34
Table 4-21: Iterations-Table (Iteration 3)	34
Table 4-22: Dependency-Table (Iteration 4)	34
Table 4-23: Time-Table (Iteration 4).....	35
Table 4-24: Free-Tasks-Table (Iteration 4).....	35
Table 4-25: To-Do-Table (Iteration 4).....	35
Table 4-26: Iterations-Table (Iteration 4)	36
Table 4-27: Dependency-Table (Iteration 5)	36
Table 4-28: Time-Table (Iteration 5).....	36
Table 4-29: Free-Tasks-Table (Iteration 5).....	36
Table 4-30: To-Do-Table (Iteration 5).....	37
Table 4-31: Iterations-Table (Iteration 5)	37
Table 4-32: Dependency-Table (Iteration 6)	37
Table 4-33: Time-Table (Iteration 6).....	37
Table 4-34: Free-Tasks-Table (Iteration 6).....	37
Table 4-35: To-Do-Table (Iteration 6).....	38
Table 4-36: Iterations-Table (Iteration 6)	38
Table 4-37: Dependency-Table (Iteration 7)	38
Table 4-38: Time-Table (Iteration 7).....	38
Table 4-39: Free-Tasks-Table (Iteration 7).....	39
Table 4-40: To-Do-Table (Iteration 7).....	39
Table 4-41: Iterations-Table (Iteration 7)	39
Table 4-42: Iterations-Table (First Experiment Result).....	40

Table 4-43: Assigning-Table (First Experiment Result)	41
Table 4-44: Iterations-Table (Second Experiment Result)	42
Table 4-45: Assigning-Table (Second Experiment Result)	42
Table 5-1: Comparison of minimum days to finish sample Sprints between SPESS and DSS tool in [21]	43
Table 5-2: Main-Table	47
Table 5-3: Main-Table (ordered)	48
Table 5-4: Main-Table (ordered)	48

LIST OF FIGURES

Figure 1-1: Overview of Scrum	2
Figure 2-1: Example – Research Methodology Input and Output	9
Figure 2-2: The Improved Scrum	10
Figure 2-3: Tasks’ Dependency Example	14
Figure 2-4: Hungarian algorithm assignment results	15
Figure 2-5: Assignment results in SPESS	15
Figure 3-1: Sprint Planning dEcision Support System (SPESS) Framework	17
Figure 4-1: Suggested Sprint Tasks’	21
Figure 4-2: Sprint Tasks’ dependency Graph	41

LIST OF ABBREVIATIONS

XP	Extreme Programming
OMG	Object Management Group
SPESS	Sprint Planning dEcision Support System
DSS	Decision Support System
TT	Total Time
LTT	Lowest Total Time
TTX	Total Time for the developer X

CHAPTER 1: INTRODUCTION

In the past decades, Agile methods have attracted the attention of software development industry and have become some of the most effective approaches due to its capability to overcome the limitations of the traditional software development approaches [1]. In [2], the authors stated that Agile methodologies are people oriented with several iterations with clients, aimed to improve customer satisfaction, intended to increase productivity and reduce risks, and used on small or large projects. Therefore, Agile has taken its place because of those characteristics which are the source of its strength [2]. There are a few Agile processes are adopted by industry, including Kanban, XP, and Scrum. The focus of this thesis is on Scrum method that is an effective method for Agile project management [3][4][5].

1.1 Scrum Overview

Scrum is a framework that provides productive and creative delivering of the products with the highest possible value, with its ability to address complex problems and fast changing requirements [6]. The Scrum model progresses via series of Sprints, which have intervals of one to four weeks and have the same length. It suggests each Sprint begin with a brief planning meeting and conclude with a review [7]. In [4], the most important benefits provided by Scrum include the greater visibility and transparency, improvement of communication between all parties, more effective teamwork, improvement of requirements capture, prioritization and planning, better focus and delivery, better management of stakeholder expectations, and finally getting greater trust and respect.

According to [8] Scrum is a framework within which people can address complex adaptive problems, while productive and creative delivering of the products of the highest possible value.

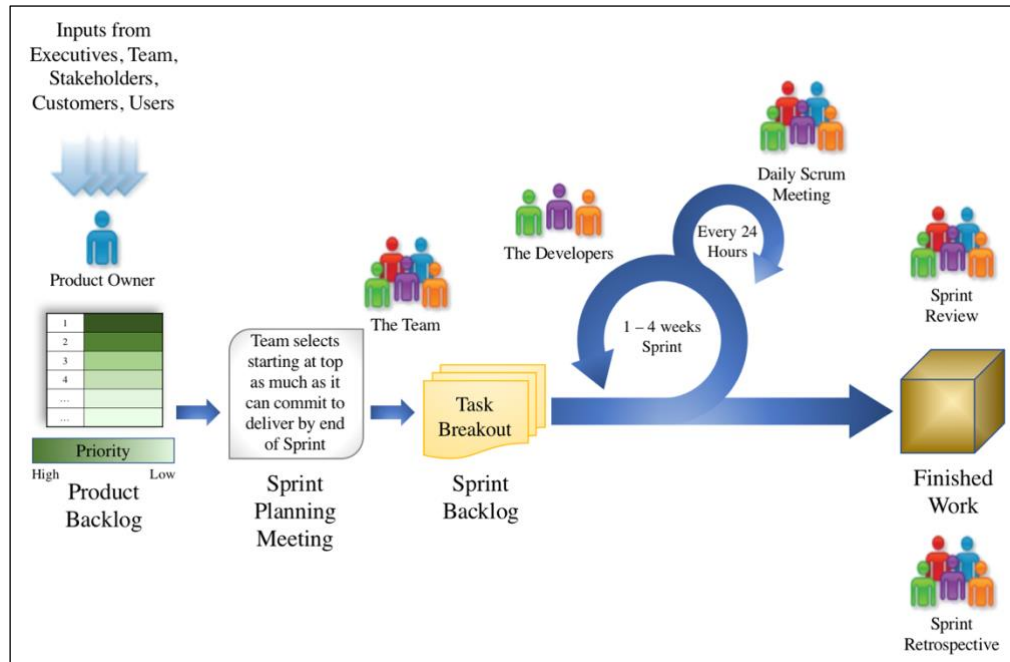


Figure 1-1: Overview of Scrum

1.1.1 Scrum Roles

This framework, Figure 1-1, is lightweight, simple to understand and difficult to master.

Scrum team consists of:

- The Product Owner is an individual (or possibly a small group) whose responsible to create a product backlog based on the product features or requirements and its priority that come from different stakeholders, or users who represent them. The Product Owner can be a product manager in a software company or other stakeholder representative or might also be a customer [9].
- The Scrum Master is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. S/he is responsible for interfacing

with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The developers team should not be considered the Scrum Master as a project manager [9].

- The Development Team is the group of highly qualified members whose responsible for developing and achieving the goals of the software product, then delivering it with the essential related documents [9].

1.1.2 Scrum Artifacts

In the Scrum process, there are two main steps, which are:

- The Product Backlog is the source of user requirements. It consists of items which may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed. The Product Backlog items is ordered in a prioritized list by the product owner who has the vision of the product that he wants to create. The items (User Stories) at the top of the Product Backlog are considered in more detail as they will be the first items to move in to the Sprint Backlog and development. There are many factors that contribute to determining the priority within the Product Backlog, such as Priority Factor, Story Size Factor and Complexity Factor [10][11][12][13]. As there are many researches that have taken place on this matter, such as [10].
- The Sprint Backlog is a set of Product Backlog items selected for the Sprint, as the Scrum framework, it starts after Sprint planning which is plan for delivering the product Increment and realizing the Sprint Goal. In sprint planning, the team makes the decision of choosing a task from the Product Backlog which they believe that they can complete it within a sprint cycle. Then come sprint backlog in which task

is broken down into units that is carried forward by the team who determines the best way to accomplish the goal within each sprint cycle [8].

1.2 Project Planning

Project planning is one of the most critical aspects of software development, thus, for Scrum as well. It determines what activities of the project needed to be done and how to be accomplished [14]. In addition, project planning provides many benefits related to management and communication [15]. As it is known, the accurate effort estimation is an essential factor for planning software projects. However, Scrum doesn't specify a unique estimation technique that assists managers to avoid problems, such as budget overruns, late delivery or lack of time, which often results in poor software quality [16][17]. Based on [17][18][19] the most used estimation technique in Scrum is Planning Poker. Planning poker, is a game that uses a consensus-based estimation technique for estimating the size of user stories and developing release and iteration plans [18].

1.3 Problem Statement

As it is known, Scrum is based on a self-organizing team which can be affected by inefficient developers who don't have the required experience for some tasks, which will reduce the team capability to carry out the tasks that meet the allocated time and quality by the organization.

1.4 Work Motivation and Objective

The main motivation for this work comes due to that increase quality and reduce time are key factors to minimize development cost of development companies and organizations which will have a beneficial influence on the customers' time and budget.

The objective of this work is to develop a method to assist the managers for assigning the tasks for the developers to guarantee that each team members contribute to the fullest of their potentials, and project planning is optimized for the shortest possible time.

1.5 Contributions

This thesis presents a Sprint Planning dEcision Support System (SPESS) which is a tool that assists the managers for Sprint planning. SPESS is mainly based on three factors: the developer competence, the developer seniority and the task dependency. This tool aims to assign the tasks of each Sprint to developers guarantee that each team members contribute to the fullest of their potentials, and project planning is optimized for the shortest possible time. The thesis contributions are as follows:

- Addressed the deficiency of previous tasks assignment practice that is not a realistic assumption of real-world practice.
- Proposed a novel tasks assignment method, namely Sprint Planning dEcision Support System (SPESS) to overcome some drawbacks of past work.
- Analyzed the time cost of using SPESS tool versus past tasks assignment practice and guaranteed that each developer has the required competence in Sprint practice.
- Incorporated tasks' dependencies into Sprint planning to ensure the optimal tasks schedule during planning.
- Evaluated the proposed approach using real and suggested Sprints scenarios.

1.6 Related Work

In [20], the authors stated that although the planning poker has many benefits, it is not an entirely efficient method yet because its result is always based on expert observations.

Therefore, they proposed identification and validation of two factors, complexity and importance of tasks, that should be taken into account by Scrum teams. The authors defined a knowledge structure that represents the effect of each factor and different aspects in the final decision. In order to have an accurate estimation for scheduling, they use a Bayesian Network to co-relate the factors which gave them the complexity of a task according to Fibonacci scale. Thus, software teams can focus on the most important tasks to obtain high quality software.

Zahraoui and Idrissi [3] stated that in Scrum projects, effort and time estimation are not sufficient to provide accurate estimates. Therefore, they adjusted story points calculation in order to produce more accurate effort and time estimate by using three factors: Priority, Size, and Complexity. Moreover, a new Adjusted Story Point measure was proposed instead of Story Point measure to calculate the total effort of a Scrum project that provides a way to use the proposed Adjusted Story Point in the Adjusted Velocity.

In [21], the authors developed a Decision Support System (DSS) tool as a hybrid methodology for an assignment of tasks, and task management. The developed tool aims to manage and optimize the time needed for developing Sprint with maintaining the inherent flexibility of Agile methodologies. In this tool, new factor—seniority—is introduced which was divided into five levels. The proposed DSS tool can assign tasks to the developers to minimize the total execution time of the Sprint. Additionally, it gave the possibility to determine whether the developers can meet the deadlines.

1.7 Thesis Organization

This thesis is organized as follows: Chapter 2 presents proposed methodology and its main attributes. The Chapter 3 describes the proposed Sprint planning tool framework. The

Chapter 4 illustrates the implementation of the methodology and shows two experiments.

Chapter 5 analyzes and discusses the results concludes the thesis and points to limitations and future work.

CHAPTER 2: THE RESEARCH METHODOLOGY

2.1 Research Methodology

In this thesis, a new Scrum tool is presented where the Planning Poker and Hungarian algorithm are adopted with more features –developer’s competence and the task dependency—in addition to the seniority factor that is used in [21]. The methodology can be formally represented as:

Objective:

- Optimize the development time with guaranteed high quality for product

Input:

- Competency Level:
 - Product Owner determines the minimum level of competence that is accepted for Sprint.
 - Developer’s self-assessed level of competency for each task.
- The dependency among tasks, if available.
- Developers’ Seniority:
 - Level of seniority for each developer.
- Estimated effort for each task based on planning poker.

Output:

- Assigning tasks for developers.

For example, Figure 2-1 shows the research methodology input and output where the input is the tasks with its information, estimated effort and dependencies, and the developers with their information, seniority level and developers' competency level for each task. Moreover, Product Owner determines the minimum level of competence that is accepted for Sprint to assign tasks. The methodology output is shown in Figure 2-1, which is assigning tasks for developers.

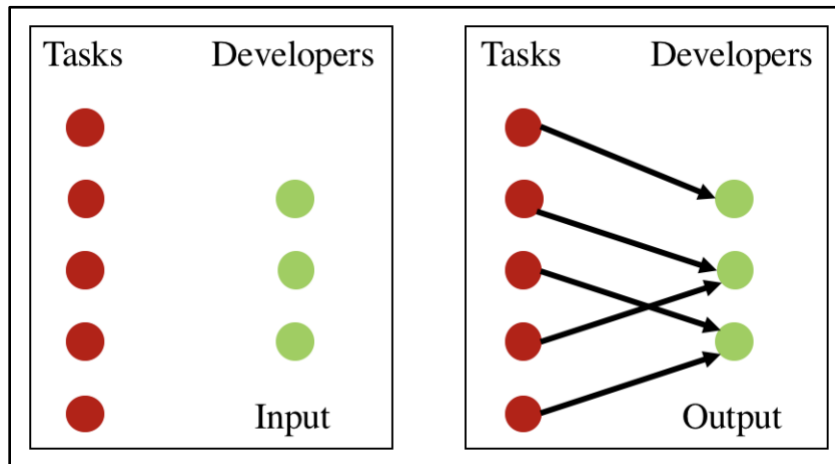


Figure 2-1: Example – Research Methodology Input and Output

In this methodology, SPESS optimizes the development time with guaranteed high quality for product.

2.2 Improved Scrum

In this thesis, the improved Scrum approach is shown in Figure 2-2, is adding new components that are Planning Poker and Sprint Planning dEcision Support System (SPESS) tool. The Improved Scrum in Figure 2-2 is as Scrum that shown in Figure 1-1, with adding new components that are Planning Poker and Sprint Planning dEcision Support System (SPESS) tool. The Planning Poker is a technique for estimating the efforts needed to develop tasks where the Sprint Planning dEcision Support System (SPESS) is a tool to

assist the managers for Sprint planning that combines the developer competency, the developer seniority and the task dependency with using the estimated efforts that comes from planning poker for each task. This tool assigns the tasks of each Sprint to developers in the Sprint, so the developers will work on their assigned tasks.

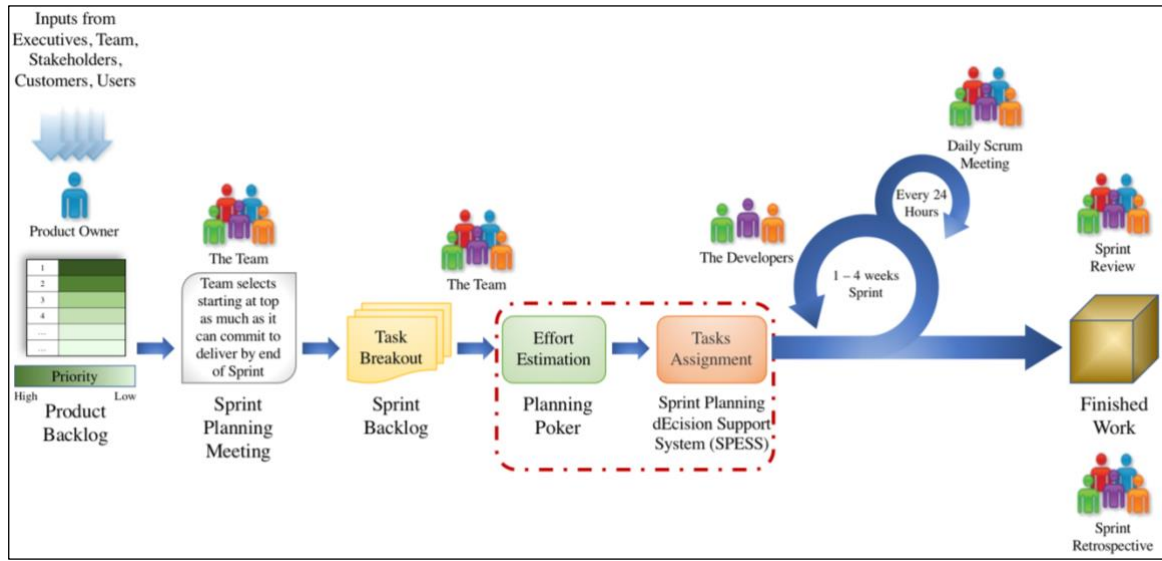


Figure 2-2: The Improved Scrum

To address the deficiencies in aforementioned Sprint planning approaches, this thesis proposes a Sprint Planning dEcision Support System (SPESS). SPESS takes into consideration additional software planning factors, including developers' competence, and tasks' dependency, The results are a more comprehensive and accurate project planning process.

2.3 Attributes of the Research Methodology

This part describes the main attributes used in SPESS method.

2.3.1 Planning Poker

Planning Poker, is a game that uses a consensus-based estimation technique for estimating the size of user stories and developing release and iteration plans [18]. It is the most used technique in Scrum projects for estimate, therefore, it is also called Scrum Poker. The

method was first defined and named by James Grenning in 2002, and later popularized by Mike Cohn in the book *Agile Estimating and Planning*. Planning Poker combines expert opinion, analogy, and disaggregation into an enjoyable approach to estimating and results in quick but reliable estimates. Participants in planning poker include all developers on the team. The Product Owner and the Scrum Master participate in Planning Poker game but do not estimate. By using Planning Poker it is assumed to get accurate estimation because all developers participate equally in the estimation process, regardless whether they are among the highest or most influential people in the Team [18]. At the start of Planning Poker process, each estimator is given a deck of cards that is based on a modified Fibonacci sequence similar to (0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100, ?, ∞, and Coffee). The Product Owner starts to read out the description of the requirements in the Product Backlog starting from the highest priority. The Product Owner and the Scrum Master answer any questions that the estimators have, then there can be a short group discussion about the concerning requirement. After all questions are answered, each estimator privately selects a card representing his or her estimate. Cards are not shown until each estimator has made a selection. At that time, all cards are simultaneously turned over and shown so that all participants can see each estimate. It is very likely at this point that the estimates will differ significantly, so if estimates differ, the high and low estimators explain their estimates. Re-estimate until the estimators have reached a loose consensus which will be the estimation size. This process should be repeated for each Product Backlog requirement.

2.3.2 The Developer's Seniority

The developer's seniority is used as proposed in [21], where there are five levels: Intern, Trainee, Junior Analyst, Analyst, and Senior Analyst. It is applied on the planning poker

to estimate effort “Poker Value” for the tasks to find the expected time to do the task. If X is the value of the estimated effort “Poker Value” through planning poker for a Task Y, the time needed for the developers to do Task Y based on their seniority will be as:

1.00X days by an Intern

0.85X days by a Trainee

0.65X days by a Junior Analyst

0.50X days by an Analyst

0.35X days by a Senior Analyst

where the day is 8 working hours.

Thus, the time schedule for (Developer X Task) is built based on the estimated effort “Poker Value” through planning poker and the developer's seniority using the following equation:

$$\text{Task Time} = \text{Estimated Effort (Poker Value)} \times \text{Seniority Level}$$

2.3.3 The Developer's Competence

Based on OMG Standard “Essence – Kernel and Language for Software Engineering Methods” [22], the competence, in general, is “a set of abilities, capabilities, attainments, knowledge and skills that are necessary to do a certain kind of work”. In this thesis’s context, the developer competency represents the abilities, knowledge and skills that the developer have to work on a specific task. The competence has five levels which, in ascending order, are (Assist, Apply, Master, Adapt, and Innovate) where Assist represents the least competence and the Innovate represents the highest competence as shown in the Table 2-1:

Table 2-1: Competence Levels and Their Checklist [22]

Competency Level	Brief Description
1 -Assists	<ul style="list-style-type: none"> - Understands the simple basics that related to this type of task. - Can follow the instructions to do the basics for this kind of tasks.
2 -Applies	<ul style="list-style-type: none"> - Is able to cooperate with others who work on this kind of task. - Is able to meet routine and simple requirements for this type of task. - Can handle simple requirements for this type of task but needs help in dealing with challenges. - Is able to think in the context and draw reasonable conclusions about the task.
3 -Masters	<ul style="list-style-type: none"> - Is able to meet most of the requirements for this type of task. - Is able to communicate and explain his or her work that related to the task. - Is able to give and receive constructive feedback related to the task. - Can identify his abilities to deal with this type of task and may need a little consultation. - Works at the professional level with little or no guidance for this type of task.
4 -Adapts	<ul style="list-style-type: none"> - Is able to meet the complex requirements for this type of task. - Is able to communicate with other developers on other tasks related to the current task. - Can direct and assist other developers in this type of task. - Is able to adapt his or her way-of-working on the task with others who are working on the same or different type of tasks.
5 -Innovates	<ul style="list-style-type: none"> - Has years of experience in dealing with this type of task. - Known as an expert in this type of tasks among his or her peers. - Supports others to work on complex tasks of this type. - Can determine when to innovate, do something different, or follow the normal procedure in the task. - Develops innovative and effective solutions to deal with the task and its challenges.

The Product Owner who is responsible for determining the minimum competence level for Sprint, will use Table 2-1 as a reference to determine the required competence level for Sprint. The developer can rank his/her competence level (1-5) for doing a specific task based on the task description and the competence levels' description in the Table 2-1.

2.3.4 The Task Dependency

The task dependency can be simply defined as the relationship between two tasks, where one task cannot start implementation until the other task is completed. One task may depend on two or more tasks, which means task must be held until all other tasks that it

depends is implemented. For example, in Figure 2-3, task_1 and task_2 can be implemented at any time without looking for other tasks, but task_3 cannot be implemented until task_1 and task_2 are completed.

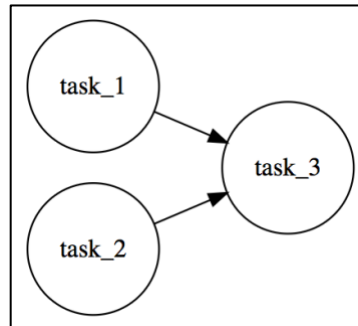


Figure 2-3: Tasks' Dependency Example

2.3.5 The Hungarian Algorithm

The Hungarian method is an optimization algorithm that combines solving the assignment problem in polynomial time and primal-dual methods. An algorithm is said to be of polynomial time if its running time is upper bounded by a polynomial expression, i.e., $T(n) = O(n^k)$ for some constant k , where primal-dual methods are a class of gradient-based algorithms for solving constrained convex optimization problems. The Hungarian method was developed and published in 1955 by Harold Kuhn. is to find the lowest overall cost of job assignments to each participant. The problem on which the algorithm applied, can be represented in a square matrix of the costs of the participants doing the jobs (Figure 2-4). In SPSS tool, Hungarian algorithm is applied in a loop of processes to handle different number of tasks and developers (Figure 2-5).

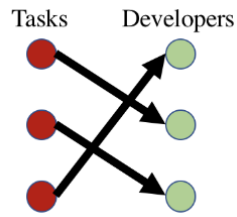


Figure 2-4: Hungarian algorithm assignment results

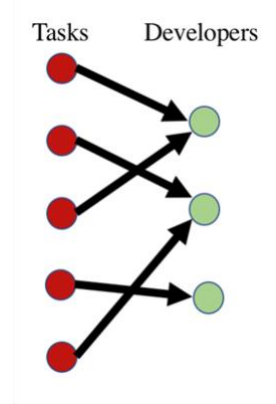


Figure 2-5: Assignment results in SPSS

CHAPTER 3: SPESS FRAMEWORK

As shown in the aforementioned literature review, most of the existing works focus on task assigning, without taking into consideration the developers' competency, which may adversely affect the delivery time and the quality of the product. For example, the tool developed by [21] is limited in dealing with the developer's working time solely based on seniority without considering developers' competence and tasks' dependencies.

Built on existing work in [21], our approach integrates two important factors -- developers' competence and tasks' dependency -- to produce a more realistic and accurate estimate of product planning management. We implement a project decision support and planning system called Sprint Planning dEcision Support System (SPESS) tool, to provide a comprehensive approach. SPESS depends mainly on its distribution of tasks on a Hungarian algorithm. The framework of SPESS tool is shown in Figure 3-1, it shows the process of the Sprint planning system which starts after determining the Sprint Backlog and finishing the planning poker meeting to find the estimated effort "Poker Value" for the Sprint's tasks.

SPESS tool is developed using PHP in XAMPP environment, JavaScript to apply a Hungarian algorithm, and MySQL database.

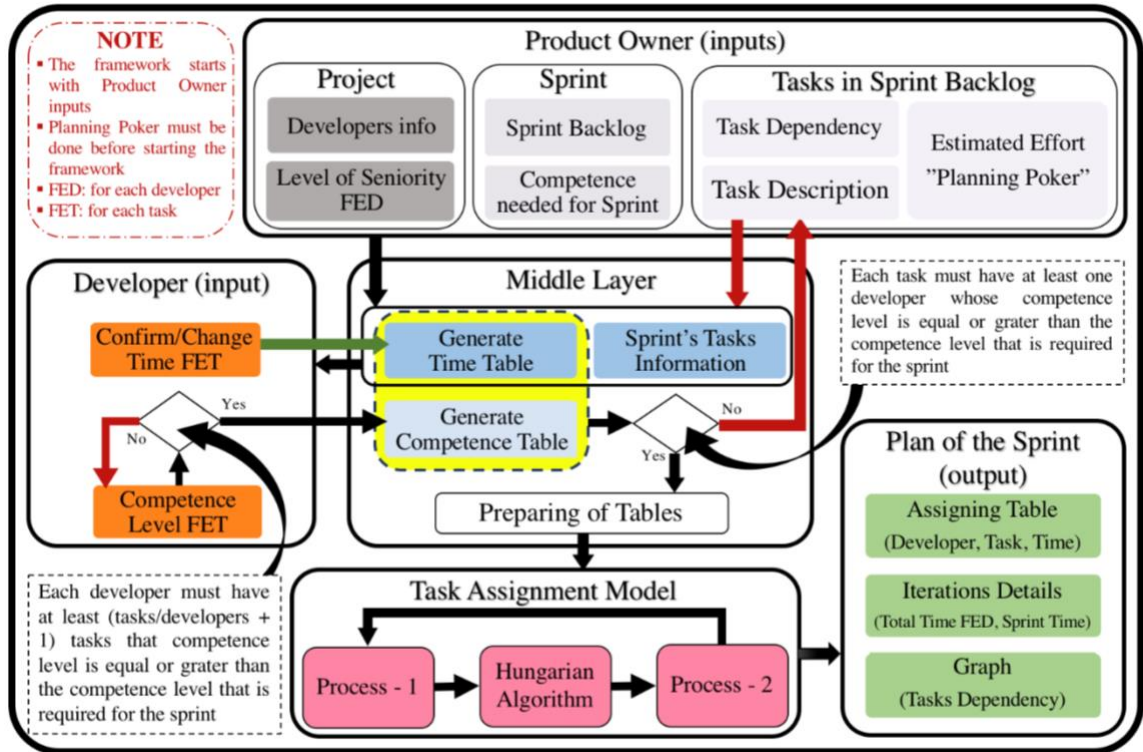


Figure 3-1: Sprint Planning dEcision Support System (SPESS) Framework

3.1 Product Owner Inputs

The SPESS framework, as it appears in the Figure 3-1, starts with the Product Owner who is responsible for the Project, Sprint, and Tasks in Sprint Backlog inputs with tasks' dependency. These inputs will be processed in the Middle Layer, where the system will generate the Time-Table which contains the developers and the time needed for doing each task based on the estimated effort and the level of seniority for each developer:

$$Task\ Time = Estimated\ Effort\ (Poker\ Value) \times Seniority\ Level$$

Then, the system will sort the tasks in Main-Table based on its estimated effort "Poker Value" and dependencies, then the Time-Table will take the same tasks order. After that the Sprint's Tasks Information with the developer's time for each task will be sent to the developers.

3.2 Developers Inputs

Each developer who is registered as a project's developer should login to the system to do his/her part of inputs. The developer will find the minimum competence level that is required for Sprint as well as Sprint's tasks with its description. Moreover, in front of each task, there will be the initial time to do the task and a competence option list from level 1 to level 5. The developer has a choice to change or confirm the suggested time, which is strongly recommended to confirm it, besides determining the competence level for each task based on the Table 2-1 levels' description. If the developer changes the suggested time that will only affect the Time-Table not the Task Poker Value. When the developer submits inputs, the system will submit the time change directly, if any, and will check if the developer has at least $(\text{tasks}/\text{developers} + 1)$ tasks that competence level are equal or greater than the required competence for Sprint or the system will give an error message for the developer to check the inputs or contact the Product Owner. The reason for using this condition is to ensure there are multiple choices for the developer even if the input is similar to another developer.

3.3 Middle Layer

The Product Owner will receive a notification email from the system when all developers have finished their inputs. Simultaneously, the Competence-Table which has the same structure and order (Developer X Task) as the Time-Table will be generated. The Time-Table contains all developers with the time they need to do each task, and the Competence-Table contains all developers' competence for each task. After that the system will check that all tasks have at least one developer whose competence level is equal to or greater than the competence that is required for Sprint. In case there are tasks that do not have any

developer with the required competence, the system will give an error message for the Product Owner with determining that tasks. Therefore, the Product Owner should have a meeting to discuss that tasks in order to improve the developers' understanding of those tasks and clarify what is unclear, then again the developers' input. In the Preparing for the Tables, the cells in the Competence-Table that have less than the required competence $C[ij]$ will be determined. Then, the value of the same cells in the Time-Table $T[ij]$ will change to be '999'. Therefore, the Hungarian algorithm will avoid these choices that have the value '999' to find the lowest cost. Unless all tasks in the iteration cannot be done because of the available developers' competence for these tasks, less than the determined competence for the Sprint, this case will be described in detail in the Illustration Chapter (Chapter 4). Moreover, the system will generate Assigning-Table and Iterations-Table for Sprint.

3.4 Task Assignment Model

The Product Owner can start Sprint processing to assign tasks for the developers using the Hungarian algorithm on the tasks' time with three processing levels.

3.4.1 Process – 1

The process – 1 will be applied on the Time-Table and Dependency-Table to generate the Free-Tasks-Table that contains tasks ready to work on by the developers. At the same process level, the system will check the available developers to determine the same number of tasks from the Free-Tasks-Table to copy them for TO-DO-Table which will be sent to the Hungarian algorithm.

3.4.2 The Hungarian Algorithm

The Hungarian algorithm will receive the TO-DO-Table as a square matrix to find the minimum time cost and send the result for the next level of the process.

3.4.3 Process – 2

In Process – 2, the results of the Hungarian algorithm will be processed to assign the tasks for the developers. In a regular case, the system will complete the assigning process for that task by copying the Hungarian algorithm result “Developer, Task, Time” to the Assigning-Table, submitting the iteration result in the Iterations-Table, and removing the Task from the TO-DO-Table, Free-Tasks-Table, Time-Table, and Dependency-Table. However, it is possible to receive “999” or “0” results, these cases are described in detail in the Illustration Chapter (Chapter 4). This is an incremental process, so the system will identify the available developers and return to work again from the first level of the processing (Process - 1) until all tasks have been assigned to get the final output.

3.5 Output

In the output, there will be the Iterations-Table that contains all developers' names and the iterations details in Sprint as well as the Assigning-Table which contains Developer, Task, and Time. Sprint will take the highest Total Time value of the last iteration from the Iterations-Table as the Sprint duration. In addition, the system will generate the Sprint Tasks' dependency Graph. Thus, developers must work on their tasks based on the iterations in the Iterations-Table to meet the Sprint Duration.

CHAPTER 4: ILLUSTRATION OF USING SPESS TOOL

This chapter illustrates in detail the application of using SPESS framework on a suggested Sprint scenario that covers all possible cases. In this experiment, in addition to the seniority factor that is used in [21], two new factors are added—developers' competence and task dependency. The suggested Sprint has 4 developers with different seniorities, where developer 1 is an Intern, developer 2 is a Trainee, developer 3 is a Junior Analyst, and developer 4 is an Analyst. In this Sprint there are 20 Tasks with different levels of dependencies as shown in Figure 4-1, which is the output Graph of the suggested Sprint.

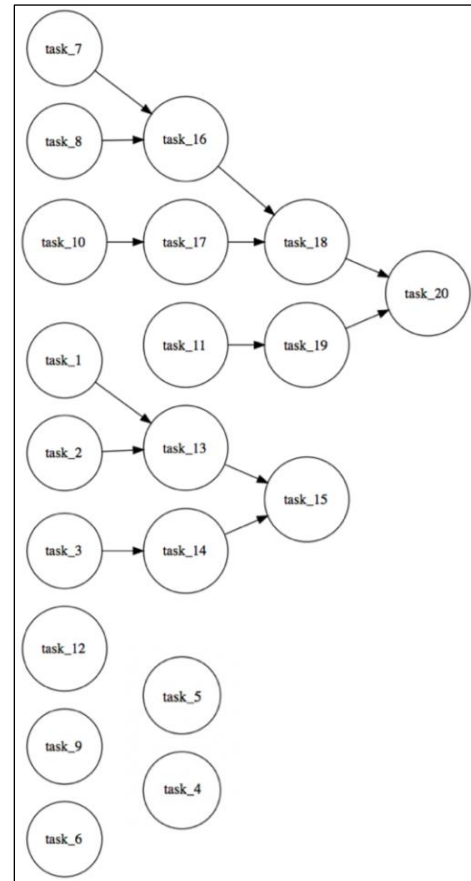


Figure 4-1: Suggested Sprint Tasks' dependency Graph

4.1 Collecting Inputs

As the framework illustrates that the Product Owner is responsible for the inputs of the Project that include the developers' information and developers' seniority (Table 4-1). For the Sprint s/he must input the Sprint backlog, and the competence needed for the Sprint (Table 4-2). In addition, s/he is responsible for input of the tasks' information, which are

tasks' description, estimated effort called "Poker Value" (Table 4-3), and tasks' dependency (Table 4-4) to the system.

Table 4-1: Developers' Seniority

Developer	Seniority Level	Seniority Value
dev 1	Intern	1
dev 2	Trainee	0.85
dev 3	Junior analyst	0.65
dev 4	Analyst	0.5

Table 4-2: Sprint Competence

The Sprint Competence	4
-----------------------	---

Table 4-3: Main-Table

Task List	Poker Value
task_1	5
task_2	8
task_3	3
task_4	5
task_5	8
task_6	3
task_7	2
task_8	5
task_9	8
task_10	1
task_11	3
task_12	2
task_13	5
task_14	3
task_15	13
task_16	2
task_17	5
task_18	5
task_19	2
task_20	13

Table 4-4: Dependency-Table

Task	Depend on	
task_13	task_1	task_2
task_14	task_3	
task_15	task_13	task_14
task_16	task_7	task_8
task_17	task_10	
task_18	task_16	task_17
task_19	task_11	
task_20	task_18	task_19

The Main-Table (Table 4-3) that contains the tasks with its estimated effort "Poker Value" will be sorted in two phases to get the ordered Main-Table (Table 4-5). The First phase is ordering the Main-Table tasks in descending order based on its Poker Value. The second phase is the reordering of the Main-Table tasks based on the Dependency-Table. For example, Task B depends on Task A, but with the sorting Task B comes first, so the system will move Task A to be before Task B. Then, the system will generate the Time-Table (Table 4-6), which contains all developers with the time they need to do each task based on task Poker Value and developers' seniority. The Time-Table will take the same task order as the Main-Table (Table 4-5).

Table 4-5: Main-Table (ordered)

Task List	Poker Value
task_10	1
task_17	5
task_8	5
task_7	2
task_16	2
task_18	5
task_11	3
task_19	2
task_20	13
task_2	8
task_1	5
task_13	5
task_3	3
task_14	3
task_15	13
task_5	8
task_9	8
task_4	5
task_6	3
task_12	2

Table 4-6: Time-Table (initial)

Time	dev 1	dev 2	dev 3	dev 4
task_10	1	0.85	0.65	0.5
task_17	5	4.25	3.25	2.5
task_8	5	4.25	3.25	2.5
task_7	2	1.7	1.3	1
task_16	2	1.7	1.3	1
task_18	5	4.25	3.25	2.5
task_11	3	2.55	1.95	1.5
task_19	2	1.7	1.3	1
task_20	13	11.1	8.45	6.5
task_2	8	6.8	5.2	4
task_1	5	4.25	3.25	2.5
task_13	5	4.25	3.25	2.5
task_3	3	2.55	1.95	1.5
task_14	3	2.55	1.95	1.5
task_15	13	11.1	8.45	6.5
task_5	8	6.8	5.2	4
task_9	8	6.8	5.2	4
task_4	5	4.25	3.25	2.5
task_6	3	2.55	1.95	1.5
task_12	2	1.7	1.3	1

After that the system will send all Sprints' task information to the developers who are responsible for inputting their competence for each task as well as confirming/changing the time needed. When the developer submits the input, the system will make sure that each developer must have at least 6 tasks (tasks/developers + 1) that competence level is equal to or greater than the sprint competence, or the system will give an error message for the developer to check inputs or contact the Product Owner.

4.2 Preparing for the Tables

After completing the inputs, the Time-Table will be changed, if need be. The Competence-Table which contains all developers' competence for each task will be generated (Table 4-7). Both Time-Table and Competence-Table will take the same structure and order

(Developer X Task) as Main-Table. Next, the system will check that all tasks have at least one developer whose competence level is equal to or greater than 4 which is the required competence for Sprint. In case, there are tasks that do not have any developer with the required competence, the system will give an error message for the Product Owner with determining that tasks. Therefore, the Product Owner should have a meeting to discuss that tasks in order to improve the developers' understanding of those tasks and clarify what is unclear, then again developers' input. Then, the system will determine the cells value in the Competence-Table that have less than level 4. After that the value of the same cells in the Time- Table will change to be '999' (Table 4-8).

In addition, the Dependency-Table will be duplicated with a new hidden table which is Reference-Table, this table will be used to find the Total Time (TT) as it will be explained later. The system will generate a table that is responsible for finding the total time for each developer in each iteration in Sprint which is the Iterations-Table. The Iterations-Table contains all developers' names and Iterations that show the iterations' assigned task and the cumulative time "Total Time (TT)". In addition, the system will generate the Assigning-Table for the Sprint that contains Developer, Task, and Time.

Table 4-7: Competence-Table

Competence	dev 1	dev 2	dev 3	dev 4
task_10	5	3	4	1
task_17	5	4	1	4
task_8	1	4	2	4
task_7	5	5	5	4
task_16	4	2	3	5
task_18	2	4	4	4
task_11	3	2	3	5
task_19	4	5	3	2
task_20	1	2	5	4
task_2	5	5	4	3
task_1	3	2	4	3
task_13	5	2	4	1
task_3	3	4	2	5
task_14	1	4	5	3
task_15	5	4	4	1
task_5	2	1	4	3
task_9	1	4	3	4
task_4	3	4	1	4
task_6	4	2	5	4
task_12	4	2	4	3

Table 4-8: Time-Table

Time	dev 1	dev 2	dev 3	dev 4
task_10	1	999	0.65	999
task_17	5	4.25	999	2.5
task_8	999	4.25	999	2.5
task_7	2	1.7	1.3	1
task_16	2	999	999	1
task_18	999	4.25	3.25	2.5
task_11	999	999	999	1.5
task_19	2	1.7	999	999
task_20	999	999	8.45	6.5
task_2	8	6.8	5.2	999
task_1	999	999	3.25	999
task_13	5	999	3.25	999
task_3	999	2.55	999	1.5
task_14	999	2.55	1.95	999
task_15	13	11.1	8.45	999
task_5	999	999	5.2	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

4.3 Process - 1

The assigning process will start when the Product Owner decides. Based on the Dependency-Table (Table 4-4), and the Time-Table (Table 4-8), the system will determine the free tasks “each task does not depend on other task(s)” and copy those tasks to a new table that is named Free-Tasks-Table (Table 4-9) with the order as the Time-Table. The Hungarian algorithm which is workable with the square matrices will be used to assign the tasks for the developers based on their time. Therefore, the system will determine tasks’ number that equal to the number of available developers, starting from the top of Free-Tasks-Table in order to apply the Hungarian algorithm to find the minimum time cost.

Table 4-9: Free-Tasks-Table (Iteration 1)

Time	dev 1	dev 2	dev 3	dev 4
task_10	1	999	0.65	999
task_8	999	4.25	999	2.5
task_7	2	1.7	1.3	1
task_11	999	999	999	1.5
task_2	8	6.8	5.2	999
task_1	999	999	3.25	999
task_3	999	2.55	999	1.5
task_5	999	999	5.2	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

For instance, suppose there are five tasks that ordered as Task_10, Task_8, Task_7, Task_11, Task_2 top-down, in the Free-Tasks-Table:

- In case there are four available developers, the system will start with Task_10, Task_8, Task_7, and Task_11 which will copy to the TO-DO-Table to apply the Hungarian algorithm.
- In case there are seven available developers, the system will copy all tasks to the TO-DO-Table with adding two tasks with zeros time to apply the Hungarian algorithm.
- In case there are five available developers, the developers' number equal the tasks' number that in Free-Tasks-Table, so all tasks will copy to the TO-DO-Table to apply the Hungarian algorithm.

4.4 Applying Hungarian Algorithm

From (Process -1) the Hungarian algorithm will receive the TO-DO-Table (iterations 1-7 in the part 4.6 Sprint's Iterations) as a square matrix to find the minimum time cost. In TO-DO-Table (iterations 1-7 in the part 4.6 Sprint's Iterations), the cells with light blue

represent algorithm results that will send for Process – 2. The orange highlight means hidden developer, so algorithm will not see these columns.

4.5 Process - 2

The Hungarian algorithm results will be processed to assign the tasks for the developers at this level. There will be three possible time values as a result “999”, “0”, or other value. Therefore, before sending the Hungarian algorithm result to the Iterations-Table and the Assigning-Table the result will be processed as follows:

- If the Time value for the Task is not “999” and not “0,” the system will complete the assigning process for that task by copying the Hungarian algorithm result “Developer, Task, Time” to the Assigning-Table, submitting the iteration result in the Iterations-Table, and removing the Task from the TO-DO-Table, Free-Tasks-Table, Time-Table, and Dependency-Table.
- If the Time value for the Task is “999” (Table 4-35– Iteration 6), the system will ignore this result for the developer (gray highlight) and remove the task from the TO-DO-Table. Then, the system will check the Free-Tasks -Table to find the next free task in the table that the developer can work on “not 999” and assign it for that developer directly (pink highlight) and complete the assigning process. If there is no task to work in, the developer will wait until the next iteration.
- If the Time value for the Task is “0” (Table 4-40 – Iteration 7), the system will ignore this result for the developer and complete the assigning process with no change to that developer Total Time (TT) in Iterations-Table from the previous iteration. Then, the system will remove the task from the TO-DO-Table because the system finds that the available developers are greater than the tasks in the Free-

Tasks-Table, so it generates this task to be able to apply the Hungarian algorithm, which is workable with the square matrices.

After assigning the iteration tasks for the developers, the duration of each assigned task will be added to the Total Time (TT) for the developer in the Iterations-Table (iterations 1-7 in the part 4.6 Sprint's Iterations). The reason for this step is to avoid the developer waiting for a long time for the next task and to try to avoid the big gaps of the total time between the developers at the end of Sprint planning. Moreover, it is the method to determine the available developer(s) for the next iteration by starting with the developer who has the Lowest Total Time (LTT), then find the other developers who their Total Time less than $LTT + 2$. Those developer(s) will be classified as “available developer” for the next iteration (send them for Process - 1). Therefore, in the TO-DO-Table the other developer(s) “not classified as available developer” information will be hidden (the orange highlight in the iterations) and the number of tasks in the TO-DO-Table will be equal to or less than the available developer(s) – based on the available tasks in the Free-Tasks-Table. In some cases, the assigned Task B for a developer depends on other Task A in one of the previous iterations, so there will be two main cases:

1. If the Task A was done by the same developer who will do the Task B – based on Iterations-Table, the Total Time for the developer will be:

$$TT \text{ (before the iteration)} + \text{Task B developer's time}$$

2. If the Task A was done by a different developer (developer X) from developer Y, who will do Task B, the system will check the Total Time for the developer X at the iteration where the Task A has been assigned (TTX), which will give two possible cases:

- i. If the TTX is less than the Total Time for the developer Y before the current iteration, the Total Time for the developer Y will be:

$$TT \text{ (before the iteration)} + \text{Task B developer Y's time}$$

- ii. If the TTX is greater than the Total Time for the developer Y before the current iteration, the Total Time for the developer Y will be:

$$TTX + \text{Task B developer Y's time}$$

- If Task B depends on more than one task, the system will compare all those tasks to find the greater TTX to do the case 2.
- The Reference-Table which will not be affected along the Sprint assigning process will be used as a reference for the dependencies in those cases.

The method is incremental, so the Process – 2 will save the result after processing. Then it will check the Time-Table if there is any task not assigned yet to back to Process – 1, or go to the final step to show the output.

4.6 Sprint's Iterations

This part shows the Sprint's Task Assignment Model processes in detail. The Product Owner will see the output of the Sprint's implementation. In TO-DO-Table (iterations 1-7), the cells with light blue represent algorithm results that will send for Process – 2. The orange highlight means hidden developer, so algorithm will not see these columns. In Time-Table (iterations 1-7), tasks with red color means that tasks depend on other task(s), and green tasks with strikethrough means that the task has already been assigned, but the developer still works on it. In Dependency-Table (iterations 1-7), The yellow highlight means that task doesn't have other tasks to depend on, so it is ready to be assigned.

4.6.1 Iteration 1

In the first iteration, Dependency-Table (Table 4-4), Time-Table (Table 4-8), and Free-Tasks-Table (Table 4-9) will be used by the system.

Available Developers:

- developer 1
- developer 2
- developer 3
- developer 4

Table 4-10: To-Do-Table (Iteration 1)

TO-DO	dev 1	dev 2	dev 3	dev 4
task_10	1	999	0.65	999
task_8	999	4.25	999	2.5
task_7	2	1.7	1.3	1
task_11	999	999	999	1.5

Table 4-11: Iterations-Table (Iteration 1)

Iterations	Iteration 1	
	Task	TT
dev1	task_10	1
dev2	task_8	4.25
dev3	task_7	1.3
dev4	task_11	1.5

4.6.2 Iteration 2

Table 4-12: Dependency-Table (Iteration 2)

Task	Depend on	
task_13	task_1	task_2
task_14	task_3	
task_15	task_13	task_14
task_16		task_8
task_17		
task_18	task_16	task_17
task_19		
task_20	task_18	task_19

Table 4-13: Time-Table (Iteration 2)

Time	dev 1	dev 2	dev 3	dev 4
task_17	5	4.25	999	2.5
task_8	999	4.25	999	2.5
task_16	2	999	999	1
task_18	999	4.25	3.25	2.5
task_19	2	1.7	999	999
task_20	999	999	8.45	6.5
task_2	8	6.8	5.2	999
task_1	999	999	3.25	999
task_13	5	999	3.25	999
task_3	999	2.55	999	1.5
task_14	999	2.55	1.95	999
task_15	13	11.1	8.45	999
task_5	999	999	5.2	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Table 4-14: Free-Tasks-Table (Iteration 2)

Time	dev 1	dev 2	dev 3	dev 4
task_17	5	4.25	999	2.5
task_19	2	1.7	999	999
task_2	8	6.8	5.2	999
task_1	999	999	3.25	999
task_3	999	2.55	999	1.5
task_5	999	999	5.2	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Available Developers:

- developer 1
- developer 3
- developer 4

Table 4-15: To-Do-Table (Iteration 2)

TO-DO	dev 1	dev 2	dev 3	dev 4
task_17	5	4.25	999	2.5
task_19	2	1.7	999	999
task_2	8	6.8	5.2	4

Table 4-16: Iterations-Table (Iteration 2)

Iterations	Iteration 1		Iteration 2	
	Task	TT	Task	TT
dev1	task_10	1	task_19	3.5
dev2	task_8	4.25		4.25
dev3	task_7	1.3	task_2	6.5
dev4	task_11	1.5	task_17	4

4.6.3 Iteration 3

Table 4-17: Dependency-Table (Iteration 3)

Task	Depend on	
task_13	task_1	task_2
task_14	task_3	
task_15	task_13	task_14
task_16		
task_18	task_16	
task_20	task_18	

Table 4-18: Time-Table (Iteration 3)

Time	dev 1	dev 2	dev 3	dev 4
task_16	2	999	999	1
task_18	999	4.25	3.25	2.5
task_20	999	999	8.45	6.5
task_2	8	6.8	5.2	999
task_1	999	999	3.25	999
task_13	5	999	3.25	999
task_3	999	2.55	999	1.5
task_14	999	2.55	1.95	999
task_15	13	11.1	8.45	999
task_5	999	999	5.2	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Table 4-19: Free-Tasks-Table (Iteration 3)

Time	dev 1	dev 2	dev 3	dev 4
task_16	2	999	999	1
task_1	999	999	3.25	999
task_3	999	2.55	999	1.5
task_5	999	999	5.2	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Available Developers:

- developer 1
- developer 2
- developer 4

Table 4-20: To-Do-Table (Iteration 3)

TO-DO	dev 1	dev 2	dev 3	dev 4
task_16	2	1.7	1.3	1
task_1	5	4.25	3.25	2.5
task_3	999	2.55	999	1.5

Table 4-21: Iterations-Table (Iteration 3)

Iterations	Iteration 1		Iteration 2		Iteration 3	
	Task	TT	Task	TT	Task	TT
dev1	task_10	1	task_19	3.5	task_16	6.25
dev2	task_8	4.25		4.25	task_3	6.8
dev3	task_7	1.3	task_2	6.5		6.5
dev4	task_11	1.5	task_17	4	task_1	6.5

4.6.4 Iteration 4

Table 4-22: Dependency-Table (Iteration 4)

Task	Depend on	
task_13		
task_14		
task_15	task_13	task_14
task_18		
task_20	task_18	

Table 4-23: Time-Table (Iteration 4)

Time	dev 1	dev 2	dev 3	dev 4
task_18	999	4.25	3.25	2.5
task_20	999	999	8.45	6.5
task_13	5	999	3.25	999
task_14	999	2.55	1.95	999
task_15	13	11.1	8.45	999
task_5	999	999	5.2	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Table 4-24: Free-Tasks-Table (Iteration 4)

Time	dev 1	dev 2	dev 3	dev 4
task_18	999	4.25	3.25	2.5
task_13	5	999	3.25	999
task_14	999	2.55	1.95	999
task_5	999	999	5.2	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Available Developers:

- developer 1
- developer 2
- developer 3
- developer 4

Table 4-25: To-Do-Table (Iteration 4)

TO-DO	dev 1	dev 2	dev 3	dev 4
task_18	999	4.25	3.25	2.5
task_13	5	999	3.25	999
task_14	999	2.55	1.95	999
task_5	999	999	5.2	999

Table 4-26: Iterations-Table (Iteration 4)

Iterations	Iteration 1		Iteration 2		Iteration 3		Iteration 4	
	Task	TT	Task	TT	Task	TT	Task	TT
dev1	task_10	1	task_19	3.5	task_16	6.25	task_13	11.5
dev2	task_8	4.25		4.25	task_3	6.8	task_14	9.35
dev3	task_7	1.3	task_2	6.5		6.5	task_5	11.7
dev4	task_11	1.5	task_17	4	task_1	6.5	task_18	9

4.6.5 Iteration 5

Table 4-27: Dependency-Table (Iteration 5)

Task	Depend on	
task_13		
task_15	task_13	
task_20		

Table 4-28: Time-Table (Iteration 5)

Time	dev 1	dev 2	dev 3	dev 4
task_20	999	999	8.45	6.5
task_13	5	999	3.25	999
task_15	13	11.1	8.45	999
task_5	999	999	5.2	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Table 4-29: Free-Tasks-Table (Iteration 5)

Time	dev 1	dev 2	dev 3	dev 4
task_20	999	999	8.45	6.5
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Available Developers:

- developer 2
- developer 4

Table 4-30: To-Do-Table (Iteration 5)

TO-DO	dev 1	dev 2	dev 3	dev 4
task_20	999	999	8.45	6.5
task_9	999	6.8	999	4

Table 4-31: Iterations-Table (Iteration 5)

Iterations	Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5	
	Task	TT	Task	TT	Task	TT	Task	TT	Task	TT
dev1	task_10	1	task_19	3.5	task_16	6.25	task_13	11.5		11.5
dev2	task_8	4.25		4.25	task_3	6.8	task_14	9.35	task_9	16.2
dev3	task_7	1.3	task_2	6.5		6.5	task_5	11.7		11.7
dev4	task_11	1.5	task_17	4	task_1	6.5	task_18	9	task_20	15.5

4.6.6 Iteration 6

Table 4-32: Dependency-Table (Iteration 6)

Task	Depend on	
task_15		
task_20		

Table 4-33: Time-Table (Iteration 6)

Time	dev 1	dev 2	dev 3	dev 4
task_20	999	999	8.45	6.5
task_15	13	11.1	8.45	999
task_9	999	6.8	999	4
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Table 4-34: Free-Tasks-Table (Iteration 6)

Time	dev 1	dev 2	dev 3	dev 4
task_15	13	11.1	8.45	999
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5
task_12	2	999	1.3	999

Available Developers:

- developer 1
- developer 3

Table 4-35: To-Do-Table (Iteration 6)

TO-DO	dev 1	dev 2	dev 3	dev 4
task_15	999	11.1	8.45	999
task_4	999	4.25	999	2.5
task_6	3	999	1.95	1.5

Table 4-36: Iterations-Table (Iteration 6)

Iterations	Iteration 1		Iteration 2		Iteration 3		Iteration 4	
	Task	TT	Task	TT	Task	TT	Task	TT
dev1	task_10	1	task_19	3.5	task_16	6.25	task_13	11.5
dev2	task_8	4.25		4.25	task_3	6.8	task_14	9.35
dev3	task_7	1.3	task_2	6.5		6.5	task_5	11.7
dev4	task_11	1.5	task_17	4	task_1	6.5	task_18	9

Iteration 5		Iteration 6	
Task	TT	Task	TT
	11.5	task_6	14.5
task_9	16.2		16.2
	11.7	task_15	20.2
task_20	15.5		15.5

4.6.7 Iteration 7

Table 4-37: Dependency-Table (Iteration 7)

Task	Depend on	
task_15		

Table 4-38: Time-Table (Iteration 7)

Time	dev 1	dev 2	dev 3	dev 4
task_15	13	11.1	8.45	999
task_4	999	4.25	999	2.5
task_12	2	999	1.3	999

Table 4-39: Free-Tasks-Table (Iteration 7)

Time	dev 1	dev 2	dev 3	dev 4
task_4	999	4.25	999	2.5
task_12	2	999	1.3	999

Available Developers:

- developer 1
- developer 2
- developer 4

Table 4-40: To-Do-Table (Iteration 7)

TO-DO	dev 1	dev 2	dev 3	dev 4
task_4	999	4.25	999	2.5
task_12	2	999	1.3	999
notask	0	0	0	0

Table 4-41: Iterations-Table (Iteration 7)

Iterations	Iteration 1		Iteration 2		Iteration 3		Iteration 4	
	Task	TT	Task	TT	Task	TT	Task	TT
dev1	task_10	1	task_19	3.5	task_16	6.25	task_13	11.5
dev2	task_8	4.25		4.25	task_3	6.8	task_14	9.35
dev3	task_7	1.3	task_2	6.5		6.5	task_5	11.7
dev4	task_11	1.5	task_17	4	task_1	6.5	task_18	9

Iteration 5		Iteration 6		Iteration 7	
Task	TT	Task	TT	Task	TT
	11.5	task_6	14.5	task_12	16.5
task_9	16.2		16.2		16.2
	11.7	task_15	20.2		20.2
task_20	15.5		15.5	task_4	18

4.7 Outputs

4.7.1 The Result of Suggested Sprint

The assigning process will continue until all tasks are assigned to the developers. When the assigning process is completed, the output of the system will be the Iterations-Table (Table 4-42) that contains all developers' names and the iterations' details in the Sprint as well as the Assigning-Table (Table 4-43) for the Sprint that contains Developer, Task, and Time. The Sprint will take the highest Total Time value of the last iteration from the Iterations-Table as the Sprint duration which is 20.2 days in our case. In addition, the system will generate Graph that shows the tasks dependencies as shown in Figure 4-2.

Table 4-42: Iterations-Table (First Experiment Result)

Iterations	Iteration 1		Iteration 2		Iteration 3		Iteration 4	
	Task	TT	Task	TT	Task	TT	Task	TT
dev1	task_10	1	task_19	3.5	task_16	6.25	task_13	11.5
dev2	task_8	4.25		4.25	task_3	6.8	task_14	9.35
dev3	task_7	1.3	task_2	6.5		6.5	task_5	11.7
dev4	task_11	1.5	task_17	4	task_1	6.5	task_18	9

Iteration 5		Iteration 6		Iteration 7	
Task	TT	Task	TT	Task	TT
	11.5	task_6	14.5	task_12	16.5
task_9	16.2		16.2		16.2
	11.7	task_15	20.2		20.2
task_20	15.5		15.5	task_4	18

Table 4-43: Assigning-Table (First Experiment Result)

Developer	Task	Time
dev 1	task_10	1
dev 1	task_19	2
dev 1	task_16	2
dev 1	task_13	5
dev 1	task_6	3
dev 1	task_12	2
dev 2	task_8	4.25
dev 2	task_3	2.55
dev 2	task_14	2.55
dev 2	task_9	6.8
dev 3	task_7	1.3
dev 3	task_2	5.2
dev 3	task_5	5.2
dev 3	task_15	8.45
dev 4	task_11	1.5
dev 4	task_17	2.5
dev 4	task_1	2.5
dev 4	task_18	2.5
dev 4	task_20	6.5
dev 4	task_4	2.5

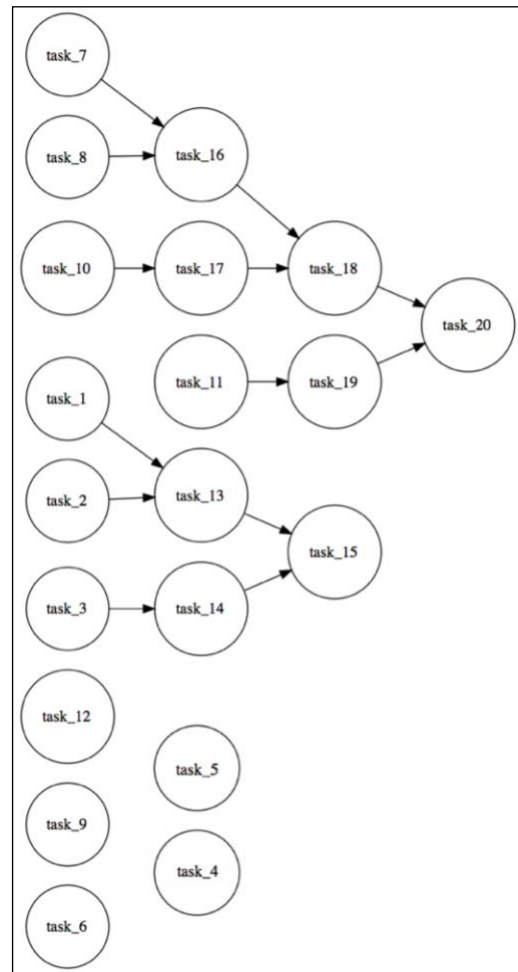


Figure 4-2: Sprint Tasks' dependency Graph

4.7.2 The Result of Real Case Study

It was occurred in a company as [21]. In this experiment there is no dependencies between tasks as well as no competence, so we apply our method on it with considering that all developers have the required competence to work on any task. This Sprint contains four developers with different seniorities, where developer 1 is a Trainee, developer 2 is a Junior Analyst, developer 3 is a Junior Analyst, and developer 4 is an Analyst. The Sprint has 16 tasks without dependencies. The outputs of the system are the Iterations-Table (Table 4-44), Assigning-Table (Table 4-45), and the Sprint duration is 17.06 days.

Table 4-44: Iterations-Table (Second Experiment Result)

Iterations	Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5	
	Task	TT	Task	TT	Task	TT	Task	TT	Task	TT
Treinee	IB11	7.00	IB1	12.00		12.00	IB8	16.00		16.00
Junior 1	IB2	5.35	IB4	9.94	IB7	13.76		13.76	IB6	15.29
Junior 2	IB14	6.88	IB9	11.47		11.47	IB3	14.53	IB15	16.06
Analyst	IB5	5.29	IB13	8.82	IB10	11.76	IB16	14.71	IB12	17.06

Table 4-45: Assigning-Table (Second Experiment Result)

Developer	Task	Time
Treinee	IB11	7.00
Treinee	IB1	5.00
Treinee	IB8	4.00
Junior 1	IB2	5.35
Junior 1	IB4	4.59
Junior 1	IB7	3.82
Junior 1	IB6	1.53
Junior 2	IB14	6.88
Junior 2	IB9	4.59
Junior 2	IB3	3.06
Junior 2	IB15	1.53
Analyst	IB5	5.29
Analyst	IB13	3.53
Analyst	IB10	2.94
Analyst	IB16	2.94
Analyst	IB12	2.35

CHAPTER 5: DISCUSSION AND CONCLUSION

5.1 Discussion, Recommendations and Limitations

5.1.1 Analysis of Results and Discussion

From the experiment results in Chapter 4, it is clear that depending solely on the factor of developer's seniority, as described in [21], is insufficient to obtain optimal Sprint time planning and maximize developers' competence. Table 5-1 shows comparison of minimum days to finish two Sprints between SPESS and DSS tool in [21] for the two experiments that in Chapter 4.

Table 5-1: Comparison of minimum days to finish sample Sprints between SPESS and DSS tool in [21]

Tools Comparison	DSS Tool in [21]	SPESS Tool
Factors Used in the Tools		
Developer's Seniority	√	√
Developer's Competency	X	√
Tasks' Dependency	X	√
Total Expected Time for Sprints Completion		
Expermint-1	Cannot be applied	20.02 days
Expermint-2	18.0 days	17.06 days

For the first experiment, the suggested Sprint that has 4 developers with different seniorities and 20 Tasks with different levels of dependencies, the proposed method in [21]

that has developers' seniority as an individual factor for assigning tasks is not workable because of the task's dependency which is not supported. Where the total expected time for completion the suggested Sprint using SPESS tool, which based on three main factors developers' seniority, developers' competency and tasks' dependency, is 20.02 days. For the second experiment, which was occurred in a company as [21], there is no dependencies between tasks as well as no required competency, so based on their proposed method [21], the expected time for completion of the Sprint in their industry case study is 18.0 days, where the result of using SPESS is 17.06 days, SPESS applied with considering that all developers have the required competence to work on any task. The reason of the difference in this particular case is due to that their method [21] relies on the equality of tasks' number assigned to each developer, which is not a realistic assumption of real-world practice. In addition to developers' seniority, SPESS also takes into consideration two additional software planning factors—developers' competency levels and tasks' dependency. Developers' competency is one of the key factors to determine the suitable developers are assigned the tasks match their skills. Task's dependency factor is used to determine the order in which the tasks are developed. Between each iteration in Sprint, SPESS checks developers working status, and prepares number of developers to work with for the next iteration. As a result, SPESS tool achieves its main objective that developers work on the tasks match their skills level with the optimal finishing time.

5.1.2 Recommendations for Using SPESS

As any system, SPESS tool should be applied in a perfect environment in terms of completeness and accuracy of Sprint's information. In this part, the most important tips are reported to achieve the best possible result using SPESS.

1. Familiarity with Customer Requirements:

Lacking of Product Owner knowledge about the customer's requirements has implications that may extend to stop the project completely. Therefore, the Product Owner must be fully sure about all the customer requirements in detail, and then arrange them according to its priority before starting the meetings of Sprint Backlog and Planning Poker.

2. Precision in Estimation:

In Planning Poker, the estimation process to find the expected effort for each task may not be accurate due to the effect of experienced developers on inexperienced developers estimates. Moreover, the estimation may be done without knowing all the requirements of the task, which may result to unacceptable results. Therefore, the team must adhere to the Planning Poker mechanism to reach the best results and to avoid the impact of developers on each other. In addition, developers must ask about unclear points before starting the estimation process.

3. Precision in Competence:

The developer may enter a competence level that is not commensurate with his/her reality, whether higher or lower, which may lead to assign task to a developer who cannot implement it correctly, which will negatively affect the time and quality of the product. Therefore, the developer must be accurate in entering competence to the system according to the table of the competences' description.

4. Precision in Tasks' Description:

The inaccuracy of the task description will affect not only the determination of the competence level and the estimated effort but also the implementation of the task. Because after assigning the task to the developer, the developer will rely on the task information

provided by the Product Owner to perform the task. Therefore, when the Product Owner input the tasks' information in the tool, s/he must write the tasks' description accurately to be a reference to the developer.

5. Following the Iterations-Table:

Failure to comply with the Iterations-Table, especially when there are tasks depend on other tasks, will have an impact on the duration of the Sprint and may exceed to the quality of the product. Therefore, it is necessary to adhere to the tasks' sequence according to the Iterations-Table of Sprint to reach the best possible results.

6. Dealing with the Change Requests:

There may be a request for modifications from the customer after starting Sprint, so the Product Owner must study the nature of the required change and its impact on the current Sprint. In some cases, it is necessary to stop the current Sprint completely and return the work again from the beginning, in other cases may be dropped some of the functions, and in other cases can be postponed until the completion of the current Sprint.

5.1.3 SPESS Limitations

As every software system, SPESS has limitations. This part reviews the SPESS tool limitations and provides proposed solutions for now or planned solutions for the future.

1. SPESS is more appropriate for planning larger number of tasks:

Certainly, SPESS tool depends mainly on the inputs of the Product Owner and the interaction of the developers which needs effort and time. Therefore, this tool becomes more effective when increasing the number of tasks, but when there are a few tasks in Sprint, the effectiveness of the tool will be inequitable comparing with the effort and the time to input adequate information.

2. SPESS requires developers have no overlap tasks among different Sprints:

Some development companies allow intersections between different Sprints, which makes developer works on other Sprint's task before completing his/her current Sprint tasks. However, SPESS tool is assumed that the developer must adhere to work in one Sprint and not work at the same time on any tasks from different Sprint until finishing all the entrusted tasks.

3. SPESS has a primitive dealing with tasks that have the Same Poker Value:

In the first experiment that described in Chapter 4, the order of tasks, in Main-Table (Table 5-2), depends on Poker Value and Dependency-Table. However, there are many tasks have the same Poker Value, but what matters most are task_20 and task_15, which have dependencies on several levels, so which task should come first?

Table 5-2: Main-Table

Task List	Poker Value
task_1	5
task_2	8
task_3	3
task_4	5
task_5	8
task_6	3
task_7	2
task_8	5
task_9	8
task_10	1
task_11	3
task_12	2
task_13	5
task_14	3
task_15	13
task_16	2
task_17	5
task_18	5
task_19	2
task_20	13

According to our experiment, the assigning process was done in the order as shown in (Table 5-3), which is considering task_20 comes first, but there is also another correct order with the same ordering mechanism, which is shown in (Table 5-4) where task_15 comes first. Therefore, when there are many tasks have same Poker Value, the ordering process will be undefined in a certain way, so the system will arrange the tasks automatically, which may lead to different results. In the experiment that described in Chapter 4, the assigning process was done based on (Table 5-3) and the Sprint Duration was (20.2), while the Sprint Duration if the assigning process was applied on (Table 5-4) would be (20.55).

Table 5-3: Main-Table (ordered)

Task List	Poker Value
task_10	1
task_17	5
task_8	5
task_7	2
task_16	2
task_18	5
task_11	3
task_19	2
task_20	13
task_2	8
task_1	5
task_13	5
task_3	3
task_14	3
task_15	13
task_5	8
task_9	8
task_4	5
task_6	3
task_12	2

Table 5-4: Main-Table (ordered)

Task List	Poker Value
task_2	8
task_1	5
task_13	5
task_3	3
task_14	3
task_15	13
task_10	1
task_17	5
task_8	5
task_7	2
task_16	2
task_18	5
task_11	3
task_19	2
task_20	13
task_5	8
task_9	8
task_4	5
task_6	3
task_12	2

To solve this problem temporarily, the Product Owner can define the tasks with the same Poker Value and arrange them by adding (0.01, 0.02, 0.03,) to the Poker Value which will affect the task order without any effect on the task time.

To solve this problem radically, SPESS tool will be improved, in the future, to determine the critical path of Sprint and find the optimal order of the tasks.

5.2 Conclusion and Future Work

5.2.1 Conclusion

Scrum is an effective Agile development method that provides fast incremental product release and quick customer feedback. Sprint planning is a critical step in Scrum practice that ensures the final product is delivered on time, within budget and with high quality. Many Sprint planning methods have been proposed, but they lack of considering of human factors (developers' competence) and tasks factors (tasks' dependency). This thesis presents a Sprint Planning dEcision Support System (SPESS) Tool. SPESS uses planning poker and Hungarian algorithm, as foundations, in addition to including seniority factor, it takes into consideration developer's competence levels and Sprint tasks dependencies. The results are a more comprehensive and accurate Sprint planning to endure fast product delivery with quality.

5.2.2 Future Work

As future work, there will have three aspects needed to add to the current SPESS Tool. The first aspect is to add a performance evaluation tool for the developers to improve the initial values that used from [21] for the developers' seniority by evaluating their history of performance on the Sprints' tasks. The second aspect is to find a suitable mechanism to determine the Product Backlog items priority. The third aspect is to expand the tool to

cover all the Scrum phases starting from the inception of the project until the final delivery of the product to the client. Once these stages are implemented, we will generalize the current Sprint Planning dEcision Support System (SPESS) to "Scrum Project Planning dEcision Support System Tool".

REFERENCES

- [1] S. Sharma and N. Hasteer, "A Comprehensive Study on State of Scrum Development," *Proceeding - IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2016*, pp. 867–872, 2016.
- [2] E. J. Quaglia and C. A. Tocantins, "Simulation Projects Management Using Scrum," *Proc. - Winter Simul. Conf.*, no. 1, pp. 3421–3430, 2011.
- [3] H. Zahraoui and M. A. Janati Idrissi, "Adjusting Story Points Calculation in Scrum Effort & Time Estimation," in *2015 10th International Conference on Intelligent Systems: Theories and Applications, SITA 2015*, 2015.
- [4] R. Kenneth, "Essential Scrum: A practical guide to the most popular agile process, Addison-Wesley," vol. 27, no. Oct, p. 427, 2012.
- [5] M. Usman, E. Mendes, F. Weidt, and R. Britto, "Effort Estimation in Agile Software Development," in *Proceedings of the 10th International Conference on Predictive Models in Software Engineering - PROMISE '14*, 2014, pp. 82–91.
- [6] K. Schwaber and J. Sutherland, "The Scrum Guide," 2011.
- [7] M. Tytkowska, A. Werner, and M. Bach, "Project Management in the Scrum Methodology," in *Communications in Computer and Information Science*, vol. 521, S. Kozielski, D. Mrozek, B. Kasprowski Paweł and Małysiak-Mrozek, and D. Kostrzewa, Eds. Cham: Springer International Publishing, 2015, pp. 483–492.
- [8] J. Sutherland and K. Schwaber, "Scrum Guides." [Online]. Available: <http://www.scrumguides.org>.

- [9] I. Sommerville, *Software Engineering*, 10th ed. 2015.
- [10] R. Popli, N. Chauhan, and H. Sharma, "Prioritising User Stories in Agile Environment," in *Proceedings of the 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques, ICICT 2014*, 2014, pp. 515–519.
- [11] K. Logue, K. McDaid, and D. Greer, "Allowing for Task Uncertainties and Dependencies in Agile Release Planning," *4th Proc. Softw. Meas. Eur. Forum*, pp. 275–284, 2007.
- [12] S. Downey and J. Sutherland, "Scrum Metrics for Hyperproductive Teams: How They Fly Like Fighter Aircraft," in *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2013, pp. 4870–4878.
- [13] A. R. Mukker, L. Singh, and A. K. Mishra, "Systematic Review of Metrics in Software Agile Projects," *COMPUSOFT, An Int. J. Adv. Comput. Technol.*, vol. 3, no. 2, pp. 533–539, 2014.
- [14] H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. 2013.
- [15] X. Wang, F. Maurer, R. Morgan, and J. Oliveira, "Tools for Supporting Distributed Agile Project Planning," in *Agility Across Time and Space: Implementing Agile Methods in Global Software Projects*, 2010, pp. 183–199.
- [16] R. Popli and N. Chauhan, "Cost and Effort Estimation in Agile Software Development," in *ICROIT 2014 - Proceedings of the 2014 International Conference on Reliability, Optimization and Information Technology*, 2014, pp. 57–61.

- [17] F. Raith, I. Richter, R. Lindermeier, and G. Klinker, "Identification of Inaccurate Effort Estimates in Agile Software Development," *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 2, pp. 67–72, 2013.
- [18] V. Mahnič and T. Hovelja, "On Using Planning Poker for Estimating User Stories," in *Journal of Systems and Software*, 2012, vol. 85, no. 9, pp. 2086–2095.
- [19] V. Mahnic, "A Case Study on Agile Estimating and Planning Using Scrum," *Elektronika ir Elektrotechnika*, no. 5, pp. 123–128, 2011.
- [20] A. Ramirez-Noriega, R. Juarez-Ramirez, R. Navarro, and J. Lopez-Martinez, "Using Bayesian Networks to Obtain the Task's Parameters for Schedule Planning in Scrum," *Proc. - 2016 4th Int. Conf. Softw. Eng. Res. Innov. CONISOFT 2016*, no. 1, pp. 167–174, 2016.
- [21] V. S. Nepomuceno and M. E. Fontana, "Decision Support System to Project Software Management," in *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, 2013, pp. 964–969.
- [22] "Essence - Kernel and Language for Software Engineering Methods," *OMG Standard*, 2015. [Online]. Available: <http://www.omg.org/spec/Essence/1.1>.