ISSN 1870-4069

Selection of Best Software Engineering Practices: A Multi-Criteria Decision Making Approach

Gil Hernández-Ledesma[1], Erik G. Ramos[2], Carlos A. Fernández-y-Fernández[2] , Jorge R. Aguilar-Cisneros[3], Juan J. Rosas-Sumano[4], Luis A. Morales-Ignacio[4]

[1] Universidad Tecnológica de la Mixteca, Ingeniería en Computación, Huajuapan de León, Oaxaca, México

[2] Universidad Tecnológica de la Mixteca, Instituto de Computación, Huajuapan de León, Oaxaca, México

[3] Universidad Popular del Estado de Puebla, Decanato de Ingenierías, Puebla, México

[4] Clínica de rehabilitación integral física y mental (CRIFYM), Oaxaca, México

hernandezlg1f@gmail.com, {erik,caff}@mixteco.utm.mx, jorge.aguilar@upaep.mx, jujorosas@hotmail.com, alfmoring@gmail.com

Abstract. The development of an application based on a set of best software engineering practices involves first their selection. This process may become complex due to the variables involved. This paper presents an Operations Research approach for the selection of best software engineering practices for the development of applications or computational systems considering a series of alternatives and selection criteria. Initially, 31 practices were considered, 19 were selected with 11 criteria using the Multi Criteria Decision Making (or MCDM) method called PAPRIKA (Potentially all pairwise rankings of all possible alternatives). This approach was applied to the development of an application that will help patients with painful hemiplegic shoulder.

Keywords: MCDM, best practices.

1 Introduction

The complexity and diversity of tasks involved in software development have resulted in methodologies that have changed according to the needs of systems and industry, the technologies involved, and scientific advances. Likewise, with the creation of development methodologies, best software engineering practices have emerged, which can be integrated into a methodology or applied jointly, indicating the steps to be taken in development.

In this paper, a MCDM method called PAPRIKA [1] was used for the selection of best software engineering practices for the development of the first component of the Hippocrates project (an application that will help patients with painful hemiplegic shoulder). An evaluation of 31 practices was done and 19 were selected using 11 evaluation criteria: C1) It can be developed efficiently by a single person, C2) Enables agile development, C3) Provides customer interaction, C4) Allows the management of progress indicators, C5) Allows early deployment, C6) Reduces risk, C7) Allows the development of a modular system, C8) Allows work in short times, C9) Allows low cost of fault repair, C10) Allows continuous testing and C11) Development team experience. Each criterion received one of the 3 possible values: bad, regular and good. According to the evaluations, the most important criterion was C1, followed by criteria C9, C5 and C10.

Hippocrates will be a system to help control people's eating habits, emphasizing their physical activity and their rehabilitation. It includes a smart assistant planner with detailed dieting habits for a patient with high cholesterol and triglyceride problems. Hippocrates components will be: A module that works with Microsoft Kinect, which will help patients with painful hemiplegic shoulder. This paper only describes the process of selecting practices for this module. A web module that will help patients keep an accurate record of what they consume and the physical activity they perform thus facilitating and improving the analysis of the nutritionist. This module will be complemented with a component that encourages you to perform physical exercise. A smart assistant which helps create diets for people with cholesterol and triglyceride problems. The remainder of the paper is organized as follows: In section two, three and four important works and concepts related MCDM and some examples of the best practices in software engineering [2] are presented; in section five, important aspects of the selection of the best practices for the development of an application that will help patients with painful hemiplegic shoulder are described; in section six, the proposed practice; in section seven the results of applying a MCDM method to the problem raised in this article. Finally, the section eight report the conclusions and propose some future work.

2 Basic Concepts of Multi-Criteria Decision Making

Multi-Criteria Decision Making (or MCDM) is an area of Operations Research, which is considered a quantitative method (these use numerical information to reach conclusions). MCDM helps to select among several alternatives from the proposed criteria for a given problem. MCDM tries to differentiate between existing alternatives and provides mechanisms to select the solution that best suits your problem. There is a great variety of MCDM methods, which can be classified in deterministic, stochastic, or Fuzzy.

Analytical Hierarchy Process (or AHP), is a method where the decision problems can be modeled with an AHP hierarchy in levels, where the attributes can be objective or subjective. This method compares alternatives through pair-wise comparisons and numerical evaluations [3]. In [4] the authors show the inconsistencies in the AHP model, so they proposed a new version of the AHP in which the relative value of each alternative was divided by the maximum of these relative values.

Technique for Order Preferences by Similarity to Ideal Solutions (or TOPSIS) is a method that uses the Euclidean distance, has problems with the consistency of the judgments, but has good results if used with the Fuzzy approach [5]. Potentially all pairwise rankings of all possible alternatives (or PAPRIKA) is a patented method and implemented in 1000Minds software. The method obtains the best alternatives through a series of comparisons among them [1].

3 Software Engineering Best Practices

A best practice in software engineering is a tool, language or methodology that represents an improvement in the development of a system or application [2]. Software engineering best practices can be classified according to their context: best practices by type of software; best practices by size of the application; best practices by activity. Some Software Engineering best practices are:

Requirements

- Inspections (requirements): This technique helps to find errors in the requirements [6].
- Product Backlog, Sprint Backlog and User Stories: The Product Backlog is the system requirements, this can change throughout the creation of the product. The Sprint Backlog are the tasks of the workers for the sprint. [7]
- Formal requirements analysis and use cases: The former is based on the creation of requirements documents with a formal notation [8] and “a use case is a unit of functionality expressed as a transaction among actors and the subject” [9].

Analysis and Design

- Unified Modeling Language (or UML) is a language that allows you to specify the elements of a software [9]. There are several UML diagrams, for example: UML class diagrams, UML Object diagrams, UML Interaction diagrams, UML State diagrams and UML Activity diagrams.
- Test-first development: This approach develops first the test cases [10].
- Simple Design: It is a design that lacks unnecessary complex elements [11].
- High-level languages: These kinds of languages have a high abstraction of computer logic.
- Object-oriented (OO) development: The object-oriented paradigm is an abstraction of programming in which everything is seen as objects and methods [2].

Development Organization

- Work organized in Sprints: A Sprint is an iteration in the development of the system or application. [7]
- Work organized in Cascade steps: This is a phase in the software lifecycle.

- Work organized in Spiral loops: A cycle of the Spiral is an iteration in which each of the activities planned for the development of the system are developed over and over again. [12]

Planning

- Planning Poker: This is a technique that uses the numbers of the poker cards to estimate the effort. There is a variation in which the numbers of the Fibonacci sequence are used. [13] [14]
- COCOMO II: This is a mathematical estimation model.

Evaluation and Control

- Gantt and PERT chart: These are charts that model the tasks in a process. [10]
- Burndown chart: This chart shows the estimated time of effort required to complete the project. [7]
- Formal progress reports (weekly): Continuous reports of the development of the system. [2]

Testing

- Formal test plans and templates: This practice uses templates with common test cases for software functions [15].
- Automated unit testing: These are tests that require minimal human intervention. [11]
- Regression test: Check that some improvement did not affect the rest of the program.
- Incremental testing Top-down: Strategy in which, the top components of the application are tested first [16].
- Incremental testing Bottom-up: Strategy in which, the terminal components of the application are tested first. [16]
- Usability test: This practice evaluates the interfaces with real users. [16]

Practices Additionally Considered

- Continuous integration: The code is integrated and tested several times a day.
- On-site customer: The customer is a fundamental part of the work team. [11]
- Prototyping: This consists of the creation of prototypes throughout the development of the system.
- Automated documentation tools: The use of these types of tools saves time and effort for the developers.
- Refactoring: This consists of making improvements to the design of the existing code. [11]

4 Related Work

The MCDM has been used in software engineering for the prioritization of requirements in [17] using the AHP method for this activity. It has also been used for the selection of a suitable software lifecycle model (SLCM), in this work [18] a fuzzy multi-criteria decision making approach is proposed.

In [19], the authors propose the selection of the best-fit agile software development methodology for small and medium enterprises based on the multi-criteria method SMARTER with a three-point scale. On the other hand, in [20] test techniques are selected using the AHP and TOPSIS methods.

5 Methodology

The application of the PAPRIKA method included in the 1000 minds software was proposed for the selection of best software engineering practices for the creation of projects that do not conform to a software development methodology. The PAPRIKA method was applied for the selection of best software engineering practices for the first component of our Hippocrates project (application that will help patients with painful hemiplegic shoulder). These are the needs that must cover the practices to be selected.

Requirements

- Partial definition of requirements: The requirements do not have to be fully defined before beginning the development of the system.
- Flexibility of requirements: The requirements may change throughout the development of the system.

Control, Evaluation, Effort Estimation and Duration of Tasks

- A method of estimating and controlling tasks is necessary.
- Tools and metrics are needed to visualize the development progress of the application.

Testing

- Test methods are executable by a single person and verify that the changes do not introduce new defects.

Additional Conditions

- No development teams are required: The component to develop is part of an individual project, so a developer will assume all roles.
- Continuous customer feedback: The client must be involved in all the application development because this component will help to provide medical treatment.

In the first place, we hoped that the development would have an agile approach, which explains why some characteristics of this approach were sought and taken. In the Requirements section, a process of requirements such as the Cascade Methodology was considered deficient, since the development process is continuous and new functional requirements can be discovered as tests or prototypes are performed. Consequently, different practices were chosen that allowed the flexibility of requirements and their partial definition.

For the section control, evaluation, effort estimation and duration of tasks, an effort estimation method was determined necessary, because this would allow the analysis and evaluation of the progress of the development of the system. In the case of testing, it was once again considered the main limitation of the system (this is an individual project), so techniques such as test automation are quite necessary. Hippocrates is a medical system and it was necessary for the clients and users (doctors) to be closely linked with its development, so that they could carry out the necessary medical validation. The practices described in the section "Software Engineering best practices" were evaluated because these fulfilled certain application needs.

The criteria considered were as follows: C1) It can be developed efficiently by a single person, C2) Enables agile development, C3) Provides customer interaction, C4) Allows the management of progress indicators, C5) Allows early deployment, C6) Reduces risk, C7) Allows the development of a modular system, C8) Allows work in short times, C9) Allows low cost of fault repair, C10) Allows continuous testing and C11) Development team experience.

An agile approach to system development was considered, due to its advantages [21]. Criterion C1 was created because this is an individual project and responds to one of the additional conditions already mentioned. Criterion C2 is used for practices with an origin in agile development to obtain a better score. Criterion C3 helped to select practices that encourage constant client intervention in development with the intention of validating the aspects related to medicine in the system. The criteria C4, C6, C7 and C9, respond entirely to the needs of control, evaluation, effort estimation and duration of tasks, since they are criteria that will help to obtain the best practices to diminish risks and to recover from errors in the development.

Table 1. Practice: Selection of best practices for software development using the MCDM method: PAPRIKA.

	Criteria												
Alternative	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	Rank	Score
Incremental testing Bottom-up	G	G	G	G	G	G	G	G	G	G	R	1st	99.8
Incremental testing Top-down	G	G	G	G	G	G	G	G	G	G	R	1st	99.8
Usability test	R	G	G	G	G	G	G	G	G	G	G	3rd	94.6
Continuous integration	G	G	-	G	G	G	G	G	G	G	G	4th	94.4
Planning poker	G	G	-	G	G	G	G	G	G	G	G	4th	94.4
Planning poker: Fibonacci	G	G	-	G	G	G	G	G	G	G	G	4th	94.4
Burndown chart	G	G	-	G	G	G	G	G	G	G	G	4th	94.4
Simple Design	G	G	-	G	G	G	G	G	G	G	R	8th	94.3
Automated unit testing	G	-	G	G	G	G	G	G	G	G	G	9th	93.3
Refactoring	G	G	-	G	G	G	G	R	G	G	R	10th	93.1
Prototyping	R	R	G	G	G	G	G	R	G	G	G	11th	91.1
PERT chart	G	G	G	G	-	G	G	G	G	G	G	12th	89.6
Work organized in Sprints	R	G	-	G	G	G	G	G	G	G	G	13th	89.1
Regression test	R	G	-	G	G	G	G	R	G	G	G	14th	88.0

	Criteria												
Alternative	C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9	C 10	C 11	Rank	Score
Product Backlog, Sprint Backlog and User Stories	R	G	G	G	G	G	G	G	G	-	G	15th	84.3
Gantt chart	G	G	-	G	-	G	G	G	G	G	G	16th	84.1
Automated documentation tools	G	G	-	R	G	G	G	G	G	-	G	17th	83.0
UML Activity diagrams	G	R	-	-	G	G	G	G	G	-	G	18th	72.0
UML class diagrams	G	R	-	-	G	G	G	G	G	-	G	18th	72.0
UML Interaction diagrams	G	R	-	-	G	G	G	G	G	-	G	18th	72.0
UML Object diagrams	G	R	-	-	G	G	G	G	G	-	G	18th	72.0
UML State diagrams	G	R	-	-	G	G	G	G	G	-	G	18th	72.0
Test-first development	B	R	R	R	G	G	G	R	G	G	B	23rd	71.9
Formal test plans and templates	B	R	-	G	R	G	G	R	G	G	R	24th	70.0
On-site customer	-	G	G	-	G	G	-	R	G	G	G	25th	67.8
Inspections (requirements)	R	R	-	G	B	G	R	R	G	R	B	26th	67.4
Work organized in Spiral loops	B	B	G	G	G	R	G	R	R	G	B	27th	62.6
COCOMO II	R	R	-	-	-	G	G	G	G	-	G	28th	56.3
Formal requirements analysis and Use cases	R	B	G	G	B	R	R	B	G	-	R	29th	53.9
Formal progress reports (weekly)	B	B	-	G	B	G	-	B	G	B	G	30th	34.3
Work organized in Cascade steps	B	B	R	G	B	B	G	B	B	B	R	31st	27.4

Criteria C5 and C8 were thought to encourage the selection of agile practices and the continuous generation of deliverables. Criterion C10 represents the need presented in the testing section. Criterion C11 allowed for the evaluation of the possible learning curve in new practices for the developer. This is important because this factor could cause variation in the development time.

6 Proposed Practice: Selection of Best Practices for Software Development Using the MCDM Method: PAPRIKA

Table 1 defines the methodology as a practice, so that it can be replicated by anyone. The practice was defined following the Essence specification, a Kernel and Language for Software Engineering Methods [22], which is a standard approved by the Object Management Group. In particular, the KUALI-BEH Kernel Extension of Essence was used, which allows for the definition of practices independently of the technology, size and life cycle of the project.

7 Results

In this project, 31 different practices for the development of the Hippocrates application were evaluated. Table 2 shows the alternatives ranked with each of their qualifications for the criteria. Hyphens (-) indicate that the alternative could not be evaluated for this criterion, the results generated by PAPRIKA were not affected by the above. The values of evaluation of the practices are: Bad (B), Regular (R) and Good (G).

The criterion C1 has the greatest weight according to the preferences detected by the method, the variation of weights for each criterion is shown graphically in Fig. 1a. The best rankings were taken as long as they were not focused on the same objectives or activities in the development of the application. The list of selected practices (grouped by type of activity) is as follows:

- Requirements: Product Backlog, Sprint Backlog and User Stories.
- Analysis and design: UML class diagrams, Simple Design, High-level languages and Object-oriented (OO) development.
- Development organization: Work organized in Sprints.
- Planning: Planning Poker Fibonacci.
- Evaluation and control: Burndown chart.
- Testing: Automated unit testing, Regression test, Incremental testing, Bottom-up and Usability test.
- Practices additionally considered: Continuous integration, On-site customer, Prototyping, Automated documentation tools and Refactoring.

Table 2. Practices evaluation for development organization.

Practice: SelectionBestPracticesForSoftwareDevelopmentUsingPAPRIKA	
Selection of best practices for software development using the MCDM method: PAPRIKA.	
Objective: Select n best practices for software development from a set of practices using the MCDM method called PAPRIKA, considering a set of criteria and needs according to the application.	
Entry. Conditions: The development team will propose a set of m practices that they themselves can develop within which n will be selected.	Result. Conditions: The development team will have a set of best practices that will fit the needs and variables of the project.
Completion Criteria: 75% of the development team members must agree to the selected practices.	
Guide	
Activity 1. Project leader will meet with the customer and discuss the needs of the application which will be listed.	
Input. Conditions: Project leader and customer are in time and place. Output. Work products: Application needs list. Competences: The project leader must have knowledge of requirements engineering. Measures: None.	
Activity 2. Development team will analyze and propose a set of criteria that reflects the application needs and other variables that they consider appropriate, for example: the type and levels of development team experience, the soft and hard skills of the developers, the type of approach to testing, organization of development, customer interaction, type of progress indicators and all those factors relevant to development.	
Input. Work products: Application needs list. Conditions: Development team is in time and place. Output. Work products: Criteria list. Competences. The development team must know or have experience in several methodologies and best practices of software development. Measures: None.	
Activity 3. The team will propose a set of practices that according to their experience could be adjusted to the needs of the project.	

Input. Work products: Application needs list. Conditions: Development team is in time and place. Output. Work products: Practices list. Competences: The development team must know or have experience in several methodologies and best practices of software development. Measures: None.
Activity 4. Give a rating (Good, Regular or Bad) to each of the practices in each of the criteria.
Input. Work products: Practice list. Conditions: Development team is in time and place. Output. Work products: Practices table with grades. Competences: The development team must know or have experience in several methodologies and best practices of software development. Measures: None.
Activity 5. Answer the questions requested by the PAPRIKA method, these will determine the preferences of the development team and reflect the needs of the project.
Input. Work products: Practice table with grades. Output. Work products: Ranking of practices. Competences. None. Measures. None.
Activity 6. Select the best ranked practices.
Input. Work products: Ranking of practices. Output. Work products: List of selected practices. Competences. None. Measures. None.

The practices with the same total score, the practices that originated in agile development (criterion C2) were chosen, due to the advantages of this approach [21]. High-level languages and Object-oriented (OO) development practices were also adopted without evaluation because the development of applications for Kinect V2 uses C#, a high-level language with the OO paradigm.

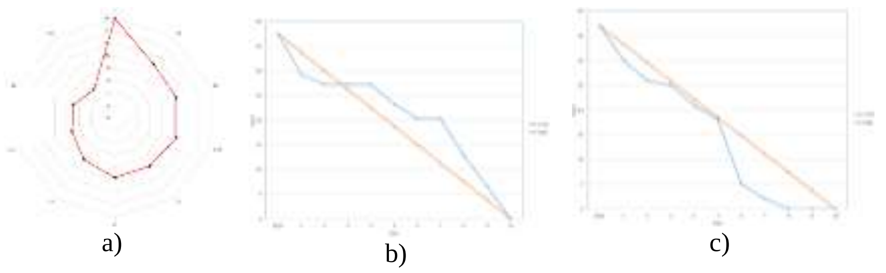

Fig. 1. a) Radar chart of criterion weights. The criterion with greater weight (15.9 %) is C1: The practice can be efficiently developed by a single person. b) Burndown chart sprint 1: It can be observed that for more than half of the sprint, the developer worked slowly but was able to complete in time all the activities planned in the sprint. c) Burndown chart sprint 2: It can be seen that an overestimation of effort was made as the goals of the sprint were fulfilled two days ahead of schedule. The blue line indicates the work to be done and the orange line the remaining ideal hours of effort.

7.1 Case Study: Best Practices Selected and the Development of the Application

The work was organized in sprints; all effort estimates were made using Planning poker Fibonacci. Below is a summary of the activities and practices used in each sprint. For the evaluation of the progress, tables were used in which the effort was documented. From these, the burndown chart of sprint 1 (see Fig. 1b) and sprint 2 (see Fig.1c) were generated.

Sprint 1: Four meetings were held with the client to create the user stories and the product backlog. The system design was analyzed and later the class diagram and the database were modeled. Three components were detected for the application: one that controls the therapists, another the patients and another the routines through which patients can be given therapy. The technologies with which the system could be implemented were analyzed.

A prototype was developed in C# for usability testing. Fig. 2a shows a screen capture of one of the prototype interfaces. A summary of the results of tests is shown below: Tests were applied to 5 different users: a physiotherapist, a physician, a psychology student with studies in industrial ergonomics, a bachelor's student and a senior citizen. Each user was chosen because their studies or condition would provide good feedback to the system that focused on senior citizens and physicians. The results revealed that: Another method was needed to remove records from patient sections, therapists, and routines; the buttons should provide feedback when the user clicks or positions on them and the silhouette (see Fig. 2b) showing the therapeutic movements to be performed had a low opacity and were difficult to perceive for the senior citizens or patients with some disease related to the hemiplegic shoulder, such as diabetes which can affect the eyes of those who suffer.

Table 3. Equivalence classes for "New Exercise" functionality.

Input Condition	Valid Equivalence Classes	Invalid Equivalence Classes
The repeats are longer than 0 digits and less than 3	length: 1-2 digits (1v)	length < 1 digit(1i), length > 2 digits (2i)
Repetitions are a number	Only has numbers (2v)	Has different characters than numbers (3i)
The degrees have a length longer than 0 digits and less than 4	length: 1-3 digits (3v)	length < 1 digits(4i), length > 3 digits (5i)
Degrees are a number	Only has numbers (4v)	Has different characters than numbers (6i)
The name of the routine is composed of numbers, letters and blanks	only has numbers, letters and blanks (5v)	has characters other than numbers, letters and blanks (7i)
The routine name has a length greater than 1 character and less than 201 characters	length: 1-200 (6v)	length < 1 (8i), length > 200

Sprint 2: In this sprint, the modules of therapists and routines were developed. Incremental testing Bottom-up was used, test cases were developed with Visual Studio software and Coded UI Tests. This tool allowed automated testing and regression tests. The strategy used includes Logic-Coverage Testing, Equivalence Partitioning and Error Guessing; Table 3 shows an example of the equivalence classes for the "New Exercise" functionality. These classes represent subsets of possible input values. The following notation was adopted: the valid classes are indicated as <Number> v, for example: 1v or 3v; while invalid classes are denoted by the letter "i", <Number> i, for example: 2i or 7i. All the code is in a Bitbucket repository with Git and code was integrated into

this repository daily once it passed the tests. The documentation was developed through Visual Studio when the methods passed the tests. Refactoring was applied to the middle of the sprint and at the end. At this time, the application is still in development. The burndown chart was an effective tool which showed the progress of the sprints and the project. This progress was in accordance with the estimates of Planning poker. Because this practice, the product and sprint backlog, user stories and planning, belong to Scrum, problems of compatibility did not arise.

The creation of test cases with the Incremental Testing Bottom-up and the regression test consumed a large part of the development time despite the use of automated unit testing. The usability tests with the prototype, provide feedback to the development of the project. In this way, errors and peculiarities that could not be described in the product backlog were detected.

With the case study, it was detected that with this approach a “methodology” or framework can be created for the development of an application, selecting the practices that best fit the characteristics of the system to be developed. The application developed in the case study is a medical system and, therefore, a greater validation by the users (doctors and patients) is necessary. This is why it is considered convenient to add a practice like the acceptance testing.

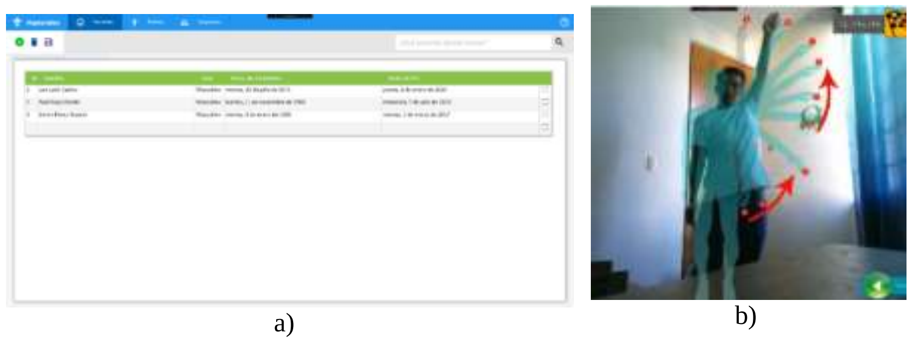

Fig. 2. a) Screenshot of the Patient module of the prototype for usability testing. b) Screenshot of the routine module (physical rehabilitation) in which it is observed that the silhouette indicating the movement is not clearly distinguishable.

8 Conclusion and Future Work

A software development can be affected by factors such as the complexity of the system and the experience and skills of those involved in the project. When using the best software engineering approach, the development may be affected by the number of selected practices and the selection criteria. Particularly if the PAPRIKA method is used, increasing the number of criteria increases the number of decisions that must be entered in the method to generate a result. In the same way, it was determined that the need for the practices be compatible. If this point is not covered, inconsistencies and possible errors will be generated.

This work differs from those already published in that in this case only one person can intervene in the project. Furthermore, different types of practices are considered to be oriented to different activities in the development of a software, not only to the testing as in [20].

If this method is not used, the developer could choose a methodology like PSP. However, works like [23] show that when this methodology is no longer mandatory, the developer leaves it. On the other hand, the developer cannot work with an agile methodology, such as Scrum or eXtreme Programming, which are based on team activities such as: Daily Scrum Meeting or pair programming. However, a methodology such as Lean UX or User-Centered Design, require multidisciplinary teams that collaborate continuously.

This is a low-medium complexity project (took 2 Sprints of two weeks each to develop a quarter of the system). The creation of this system cannot be guided solely by the intuition of the developer, but rather requires a methodology or framework because in its absence, there will be no organization for approximately 4 months of work.

This methodology allows you to select software engineering practices and reduce the possibility of the developer falling into bad practices such as: not documenting code or not fully testing all the features. Furthermore, by using this approach one can adapt or "create" a methodology for unusual projects or constraints like time, money, the experience of developers, as there will not always be ideal projects and clients who are willing to invest money or the time of your staff to perform, for example, continuous iterations of prototypes or usability tests.

The selection of practices from agile methodologies was a success, since these were compatible and adaptable to the main limitation of the system (a developer will assume all the roles), although some practices were not considered for this reason, for example, in a pair programming or Scrum daily meeting. The development times coincided with the estimates, the objectives were reached and the expected products were generated. In general, the practices helped positively in the development. The only drawback was the testing strategy, since the developer did not have any experience in this area, which led to this task consuming more time than expected.

If a Project Leader made this selection of practices, the human factor would be involved. This would be an empirical process in which errors might occur because each project has different needs, different technologies, different work teams with a certain type and levels of experience and hard and soft skills. All of the above factors can result in a large number of variables to be considered, which can be easily represented in a MCDM method.

This article uses an approach to create an application development framework based on the selection of the best practices that adapt to the circumstances of the teamwork, delivery times, the developer’s experience, the tools available and all those criteria that the project manager, technical leader or developers consider important. Future work would involve using this approach with a greater amount of initial software engineering practices, developing an analysis of the compatibility of practices and using other methods of MCDM for selection.

References

1. Hansen, P., Ombler, F.: A new method for scoring additive multi-attribute value models using pairwise rankings of alternatives. Journal of Multi-Criteria Decision Analysis, 15(3-4), pp. 87–107 (2008)
2. Jones, C.: Software Engineering Best Practices. 1st edn., McGraw-Hill, Inc., New York, NY, USA (2010)
3. Majumder, M.: Impact of urbanization on water shortage in face of climatic aberrations. 1st edn., Springer Singapore (2015)
4. Belton, V., Gear, T.: On a short-coming of Saaty's method of analytic hierarchies. Omega 11(3), pp. 228–230 (1983)
5. Velasquez, M., Hester, P. T.: An analysis of multi-criteria decision making methods. International Journal of Operations Research, 10(2), pp. 56–66 (2013)
6. Shull, F., Rus, I., Basili, V.: How perspective-based reading can improve requirements inspections. Computer, 33(7), pp. 73–79 (2000)
7. Schwaber, K.: Agile project management with Scrum. Microsoft press (2004)
8. Ciancarini, P., Cimato, S., Mascolo, C.: Engineering formal requirements: An analysis and testing method for z documents. Annals of Software Engineering, 3(1), pp. 189–219 (1997)
9. Rumbaugh, J., Jacobson, I., Booch, G.: Unified modeling language reference manual. 2nd edn., Pearson Higher Education (2004)
10. Lethbridge, T., Laganiere, R.: Object-Oriented Software Engineering: Practical Software Development Using UML and Java. McGraw-Hill, Inc., 1st edn., New York, NY, USA (2002)
11. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change. 2nd edn., Addison-Wesley Professional (2004)
12. Sommerville, I.: Software Engineering. 8th edn., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2006)
13. Mahnic, V., Hovelja, T.: On using planning poker for estimating user stories. Journal of Systems and Software, 85(9), pp. 2086–2095 (2012)
14. Tamrakar, R., Jorgensen, M.: Does the use of Fibonacci numbers in planning poker affect effort estimates? In: 16th International Conference on Evaluation & Assessment in Software Engineering, EASE 2012, Ciudad Real, Spain, May 14-15, 2012, pp. 228–232 (2012)
15. Nguyen, H. Q.: Testing applications on the Web: Test planning for Internet-based systems. John Wiley & Sons (2001)
16. Myers, G. J., Sandler, C.: The Art of Software Testing. John Wiley & Sons (2004)
17. Karlsson, J.: Software requirements prioritizing. In: Proceedings of the Second International Conference on Requirements Engineering, pp. 110–116 (1996)
18. Hicdurmaz, M.: A fuzzy multi criteria decision making approach to software life cycle model selection. In: 38th Euromicro Conference on Software Engineering and Advanced Applications, pp. 384–391 (2012)
19. Silva, V. B. S., Schramm, F., Damasceno, A. C.: A multicriteria approach for selection of agile methodologies in software development projects. In: IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 2056–2060 (2016)
20. Victor, M., Upadhyay, N.: Selection of Software Testing Technique: A Multi Criteria Decision Making Approach. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 453–462 (2011)

21. Petersen, K., Wohlin, C.: A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. Journal of systems and software 82(9), 1479–1490 (2009)
22. OMG: Essence - kernel and language for software engineering methods. Available at : http://www.omg.org/spec/Essence (2015)
23. Johnson, P. M., Kou, H., Agustin, J., Chan, C., Moore, C., Miglani, J., Doane, W. E.: Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In: Proceedings of the 25th international Conference on Software Engineering, IEEE Computer Society, pp. 641–646 (2003)