Identifying Success Factors in a Legacy Systems Reengineering Project Using Agile Methods

Everton Mateus Fernandes(✉) and Thiago Schumacher Barcelos

Laboratório de Computação Aplicada – LABCOM3, Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Guarulhos, São Paulo, Brazil
everton.mfernandes@gmail.com, tsbarcelos@ifsp.edu.br

Abstract. System maintenance or extension costs, during its lifecycle, can exceed the cost of its rewriting and can lead companies to choose a reengineering strategy. On the other hand, several similarities between recommended practices for reengineering projects and agile practices can be found in the literature. Hence, this article aims to understand the success factors that influence the reengineering process of legacy systems and how agile methods can influence the results. A real project of a software development company in the city of Sao Paulo was used as basis for a case study of a legacy system reengineering to a SOA application. The project results were compared with the perception of the development team through semi-structured interviews, the analysis of project artifacts and the best practices proposed in the literature to understand whether the results mentioned in the literature would be confirmed in practice. The obtained results reinforce the hypothesis that reengineering projects can be more successful when developed using agile methodologies.

Keywords: Software reengineering · Agile methods

1 Introduction

Legacy software systems are maintained as long as their maintenance and evolution costs outweigh the replacement or rewriting costs. The cost of maintaining or extending a system can increase due to several factors, but we can highlight architectural, documentation, design or granularity problems of its components [1, 2]. All of these factors over time can lead to a high cost of maintenance or impossibility to implement extensions and, therefore, lead to the decision to abandon the system, rewrite it or reengineer it.

The reengineering process involves understanding a legacy system and redeploying its functionality in order to improve the quality of functional and non-functional requirements [1]. The reasons for adopting a system reengineering approach can be diverse as time/cost to create a new system, knowledge added to the existing product, adherence to the company's business, among others [2]. Thus, the system reengineering process aims to rebuild the system in a new form to make its maintenance and extension costs more sustainable.

P. Meirelles et al. (Eds.): WBMA 2019, CCIS 1106, pp. 101–110, 2019.
https://doi.org/10.1007/978-3-030-36701-5_9

The system reengineering process can be done iteratively, going through cycles of requirements gathering, risk assessment, new system engineering and results evaluation [1, 3]. In addition to the iterative and incremental cycle, several standards are proposed in the literature for this process [1–3].

Iterative and incremental development processes have gained strength in the industry with the adoption of agile project management and system development methodologies. The main gains of these methodologies are the ability to deliver continuous value, flexibility to change, increased confidence in code through automated testing, among others [4].

In this article we present the results of a reengineering project of part of a legacy Enterprise Resource Planning (ERP) system, developed and distributed by a software development company based on São Paulo, Brazil as a case study to understand how the reengineering practices and agile methodologies proposed in the literature affected the project outcome. A literature review was conducted to find reported best practices and success factors for both reengineering projects and projects using agile methods. In Sect. 2, related works and main concepts applied in the study are presented. In order to understand the application of the practices proposed in the literature, Sect. 3 will detail the project, then Sect. 4 will present the interview data, Sect. 5 will show an analysis of documental data and Sect. 6 will discuss project results through triangulation between literature data, team perceptions obtained through interviews and document analysis. Finally, Sect. 7 will present the conclusions of the study.

2 Related Works

2.1 Systems Reengineering

The reengineering process involves understanding a legacy system and redeploying its functionality to improve functional and non-functional requirements [1]. To achieve the expected results, the legacy system goes through the reverse engineering processes of the application and subsequent reimplementation of its requirements.

Khadka *et al.* [5] searched for reengineering models in the literature and found variations of Plan, Do, Check and Act (PDCA) software development models focused on the reengineering process. This process works in an iterative way, starting with understanding the legacy software, designing the intended software, performing a feasibility study and then going through cycles of choosing and applying migration, implementation and deployment techniques.

Reengineering projects have factors in common which have been identified as success factors raised from case studies [6]. Among them we can highlight:

- Legacy system potential: Complexity, documentation, testing, code quality, and access to undocumented legacy information influence the reengineering process.
- Migration strategy: The strategy should be chosen taking into consideration financial, technical and people resources. Techniques include covering old code with new services, rewriting the logic of a monolithic system into smaller services, among others.

- Company business process: Company engagement for the project is very important, as usually some stakeholders are at the top hierarchical levels and have the necessary influence to drive the project forward. Therefore it is important for reengineering to be aligned with business needs to show value in the proposed deliverables and not just rewrite the code.
- Budget and Resources: Depending on the size of the legacy, proposed systems change, and business impact, the reengineering process can take more or less time, and financial, technical, and human resources are keys to give body and structure to the continuity of the project.
- Constant monitoring of the migration process: Monitoring committees and periodic process reviews help keep the project course in line with company expectations and even allow direction changes when needed.
- Testing: In the legacy system they help to document functionality and in the new application they help to validate and guarantee deliveries in terms of performance, reliability and safety.
- Team technical skills: The team's technical level can directly impact on time and quality of delivery, since it may be necessary for the team to acquire new knowledge and expertise during the project.

2.2 Agile Methods, Practices, and Success Factors

Melo *et al*. [7] and Mazuco [8] identified through structured interviews the most widely used agile methods in the Brazilian software industry and listed Scrum and a mixed version of Scrum and XP as the most commonly used methodologies and daily meeting, unit testing, sprint planning, product backlog and release planning as the most adopted practices. In the mentioned works there is also consensus that the adoption of agile practices brings improvements such as increased productivity, better adaptability to changes, improved team communication and increased quality of delivered software.

Through a systematic literature review [9] 14 factors that influence the success of projects using agile methods were identified. These factors were divided into 3 categories and can be seen in Table 1.

Table 1. Factors that influence the success of agile projects

People	Processes	Technology
Competence and Expertise	Agile Practices	Appropriate Technical Training
Executive Support	Deliver key functionality first	Tests (Unit, Integration, etc.)
Team and user motivation	The right amount of documentation	Simple Design
Small Teams	Strong Communication	Tooling Support
User Participation		Well-defined coding standards

3 Case Study

In order to confront the practices proposed in the literature with a real application, this article gathered data from a reengineering project of part of a legacy Enterprise Resource Planning (ERP) system, developed and distributed by a software development company based on São Paulo, Brazil. The case study was chosen as the research strategy; according to Wholin et al. [10], this strategy is suitable for studying phenomena in real contexts and also for confronting the obtained results with earlier studies or theories. The same authors argue that analysis of multiple sources of information is important to ensure the validity of the results. Hence, for project analysis, a triangulation strategy [11] was used to confront literature data, documental analysis of project artifacts (Jira, Git, etc.) and semi-structured interviews with those involved in the development phase.

The company's legacy product consisted of a Client × Server application and a web portal. The solution could be installed in a customer-owned environment (On Premise), in a cloud provider infrastructure as a service (IaaS), or through software as a service (SaaS) model provided by the software company. The backend and web services used on the legacy system's web portal were written in the company's proprietary language using a structured paradigm. Application code was shared across the entire ERP system, which made it difficult to scale specific services. The frontend mainly used pure Javascript with Bootstrap for validations and building interface components.

After analyzing the difficulties generated by the complexity of the legacy product, the company compared the costs involved and potential benefits of legacy maintenance and chose to reengineer part of the application by creating a new product based on a fully service oriented architecture (SOA), offering it only as SaaS hosted by a third-party vendor. The new product was still written in the same proprietary language but using the object-oriented paradigm. Specifically, standalone ERP-isolated services were developed, providing individual scalability. Webservices stopped using SOAP and started using REST following OpenAPI standards. For the frontend a framework based on Angular 2 was used and the interface was completely redesigned to improve usability.

The expected gains at the start of the project were: improved code quality; a more robust system architecture, creating an application that is easier to administer and deploy; decreased concurrency between application functionalities, and to create a independent product that could be offered without the need to purchase the entire ERP.

The team was initially composed of 3 middle level developers, 1 Product Owner and was subordinate to 1 product manager. At the time of writing, the team had 5 developers, 1 DevOps analyst, 1 Tester, 1 Scrum Master and was under duties of 1 product manager and 1 engineering manager. During the project 3 developers were relocated to other demands. The project started in July 2017 and lasted about 20 months, being one month to initial understanding of the problem and 19 months to development. The initial understanding phase served to understand the project needs and validate with customers whether the proposed solution would be appropriate to them. For that, a Design Sprint week was held, high fidelity prototypes were created and the minimum viable product (MVP) validated with the customers. In the development phase, initial Sprints 1 through 5 were used to understand the legacy software code, design the classes that would be used as the basis for the backend and to design and develop interfaces based on validated prototypes with customers. From Sprint 6 on the features development began. At Sprint 13, with the

input of a DevOps analyst, the production environment began to be prepared to receive the application. Following the start of testing in an environment nearest to production, defects and corrections began to be recorded from Sprint 19. The project matured and at Sprint 26, which took place in November 2017, the pilot project was carried out in first customer. With the pilot's success the system went into production and new features continued to be implemented in the product. New integrations and maintenance actions were also registered.

4 Interviews

After 35 project sprints a semi-structured interview was conducted through an online form with some of those involved in the development phase to understand the agile and reengineering practices used and their impacts on the project from the team's point of view. Then, the content was grouped by similarity of themes using content analysis [11]. The interviews were answered by the Product Owner and 4 developers. The main questions are listed in Table 2, and the full content of the interviews can be accessed at http://bit.ly/2lvUZDZ. Nine recurring themes were identified in the interviews, which are presented below in Table 3.

Table 2. Main questions of the interview.

Q1 - What strategy was adopted to revitalize the legacy system?
Q2 - How good is the legacy system architecture?
Q3 - Did the team have all the skills and technical knowledge at the beginning of the project? How was the knowledge acquisition during the project?
Q4 - How was the project management support?
Q5 - How was the pilot user accepting the project idea and after implementation?
Q6 - How did the team grow or shrink during the project? How did this impact throughout the process?
Q7 - How was the user participation in the project decision making?
Q8 - Which agile practices have been adopted?
Q9 - How did agile practices contribute to the success of the project?
Q10 - How the application was documented? Is there documentation requested by the customer?
Q12 - How did team communication influence the project?
Q13 - What types of tests were applied? Has this practice given safety and quality to the product?
Q14 - How complex is the design of the end application?
Q15 - What is the toolset used in the application development, maintenance and deployment process? How did this tooling evolve?
Q16 - Are coding standards defined? How does this help in the project?

Table 3. Themes found in interviews

Theme	Citation examples
Learning throughout the project	Learning a new framework
Legacy issues and difficulties	Complex code, lack of documentation
Legacy strengths	Business rule adherence
Using reengineering practices	Legacy code study, rules extraction
Use of agile practices	Sprint planning, pair programming
Good management support	Team autonomy
Project difficulties	New developers adapt
Project gains	More stable product

Agile practices were well accepted by the team and were perceived positively. As key benefits the team cited improved communication, close contact with the customer, functionality deliveries that add customer value. The most mentioned practices are presented below in Table 4.

Table 4. Most mentioned agile practices

Architectural spikes/spike solutions	Retrospectives
Version control	Demonstration or review meeting
Definition of done	Daily meeting
Frequent deliveries	Code review
Product backlog	Scrum master
Task board	Sprint backlog
Refactoring	TDD

The use of a proprietary programming language and the adoption of the Object Oriented paradigm were seen as an initial hindering factor for learning; on the other hand, coding standards and automated test cases were pointed out as facilitators of knowledge transmission and helped, in the team's perception, to reduce the time between the developer's entry and the start of his work on the project. In addition, the team had to deal with a completely new architecture by company standards. This required several stories of study.

The team considered that the product was easy and intuitive for the end user, due to the fact that, without training, after the pilot, about 200 users were accessing the product in a fluid manner and with no maintenance usability issues.

Another perception of the team is that now it is easier to implement new features and maintain the product. Automated testing brought more security to perform code maintenance and refactoring. Serious production problems are rarely encountered and

the tests help to guarantee that the new sprint deploy won't break the code or process in production.

5 Document Analysis

As an indicator of the evolution of team deliveries we used the number of commits performed per month, shown in Fig. 1, and the amount of points delivered by Sprint presented in Fig. 2.

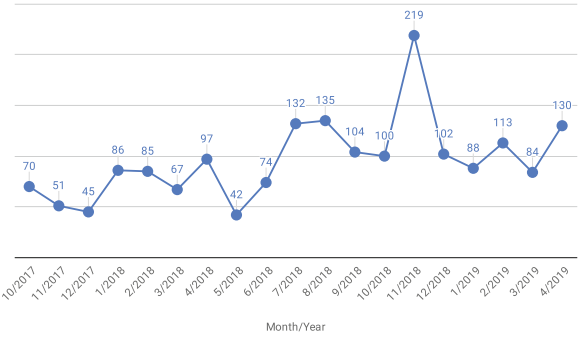

Fig. 1. Commits by month/year

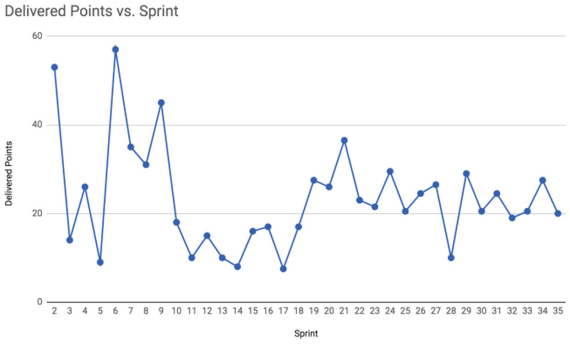

Fig. 2. Delivered Points vs Sprint

We gathered data about the movement of team members with the product owner of the project. Movement data included active people who joined and left the project and people who were on vacation. The graph showing this movement is shown in Fig. 3.

Figure 4 shows the evolution of the number of test cases. The quantity of test cases present at the end of each sprint was gathered from the version control repository.

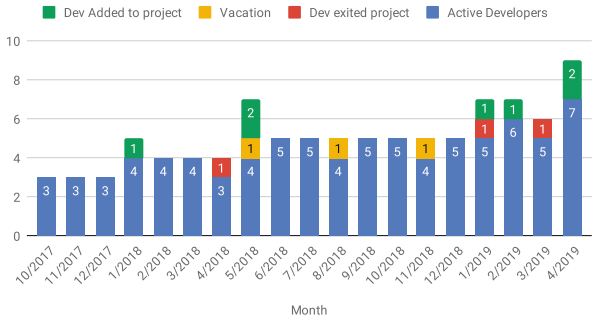

Fig. 3. Staff variation per month

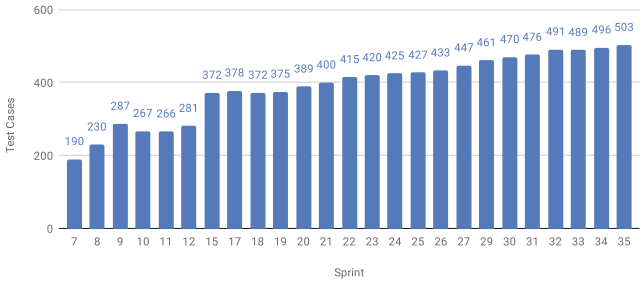

Fig. 4. Number of test cases per Sprint

6 Analysis of Results

In the test case chart (Fig. 4) we can notice that developers adopted the creation of automated testing as a practice and continued to develop new test cases throughout the project. This practice was cited in the interviews as a positive factor; besides this, it is a recommended practice for both reengineering [6] and agile methods [9].

In some interview excerpts it was mentioned that the change and departure of developers affected the project progress, this can be reinforced by comparing the staff variation chart by month (Fig. 3) with the commit chart by month (Fig. 1), where there is a decrease in the number of commits in the months when developers left or when a developer went on vacation. It is worth noting that the commit chart per month (Fig. 1) tends to return to the previous level in the month following the staff exit, thus suggesting that a new developer adaptation time tends to last about one month. This was considered a short time by the team. The short adaptation time may have contributed to keeping the number of points delivered by mid-sprints to the end of the project stable even with 4 new developers coming in and 2 leaving in the last 3 months of the project as seen in Fig. 2.

In the commit chart by month (Fig. 1) it is possible to observe a peak of commits in November that coincides with the pilot project showing that the team had a different approach during this period to support the demands of the project in production, but soon the demand normalized, thus suggesting that the project in production remains stable and

does not have high demands for maintenance, a suggestion also made by those involved in the interviews.

Based on the practices proposed in the literature, we can identify common factors between software reengineering and agile practices. Technical expertise, testing, management support and simple design are factors mentioned in both methodologies and were present in this reengineering project according to the interviewees. In this project, we could see many factors that helped the good results indicated by the team. Some of them are: choosing a migration strategy in initial sprints to support the knowledge extraction from legacy code; the presence of executive support, budget and resources to execute the project and alignment with the company's business process; constant monitoring of the migration process with scrum ceremonies and agile practices; applying testing practices from the beginning; team and user motivation with constant user participation; organization of small teams; delivering of key functionality first; producing only the necessary amount of documentation; usage of strong communication to align expectations and learn from each other; providing technical training during the project; producing a simple design; usage of tooling support and well-defined coding standards. Despite that, some difficulties were also found because the legacy system was hard to maintain, extract information and test, did not have automated tests or good documentation. However, the migration strategy was effective enough to get the knowledge from the source code. The competence and expertise had to be obtained during the processed supported by agile practices, and this made the project run slower than expected.

7 Conclusions

Given the similarity between a subset of successful practices in agile and reengineering projects, as well as the suggestion to organize reengineering processes in an iterative and incremental fashion, it is valid to suppose that reengineering projects tend to be most successful when developed using agile methodologies. Thus, it is also possible to conclude that the success factors of agile projects, such as technical expertise, simple design, testing, among others, can also positively affect reengineering projects. This hypothesis is preliminarily supported by case studies such as that presented by Galinium and Shahbaz [6], which identified a demand for high level of communication, visibility of progress and adaptation to change when necessary in reengineering processes. In the case study presented in this article, based on the analyzed data we found evidence that agile methods in fact supported the execution of the reengineering strategy, as they allowed the team to acquire and disseminate knowledge throughout the project through spike practices, team lectures for team and pair programming. In addition, agile practices served to ensure that business and customer needs were met from the outset of the project and that customer demands were constantly prioritized.

The reengineering project was considered successful from the customer's point of view, as identified through testimonials given to the development team and mentioned in the interviews. The reengineered system provided better performance and more stability. From the development team's point of view, the application now has better code quality and easier maintenance and development of new features. For the company, the developed product is aligned with its strategy and meets the needs of its customer. The project results

helps to reinforce the hypothesis that reengineering projects can be more successful when developed using agile methodologies.

References

1. Demeyer, S., Ducasse, S., Nierstrasz, O.: Reengineering patterns. In: Object-Oriented Reengineering Patterns, pp. 1–14. Elsevier (2003). https://doi.org/10.1016/B978-155860639-5/50006-7
2. Majthoub, M., Qutqui, M.H., Odeh, Y.: Software re-engineering: an overview. In: 2018 8th International Conference on Computer Science and Information Technology (CSIT), Amman, pp. 266–270. IEEE (2018). https://doi.org/10.1109/CSIT.2018.8486173
3. Sahoo, A., Kung, D., Gupta, S.: An agile methodology for reengineering object-oriented software. Presented at the 28th International Conference on Software Engineering and Knowledge Engineering, 1 July 2016. https://doi.org/10.18293/SEKE2016-227
4. 13th Annual State of Agile Report. https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508
5. Khadka, R., Saeidi, A., Idu, A., Jurrian, H., Jansen, S.: Legacy to SOA Evolution: A Systematic Literature Review. IGI Global, Hershey (2012)
6. Galinium, M., Shahbaz, N.: Success factors model: case studies in the migration of legacy systems to Service Oriented Architecture. In: 2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE), Bangkok, Thailand, pp. 236–241. IEEE (2012). https://doi.org/10.1109/JCSSE.2012.6261958
7. de O. Melo, C., et al.: The evolution of agile software development in Brazil: education, research, and the state-of-the-practice. J. Braz. Comput. Soc. **19**, 523–552 (2013). https://doi.org/10.1007/s13173-013-0114-x
8. Mazuco, A.S. da C.: Percepções de Práticas Ágeis em Desenvolvimento de Software: Benefícios e Desafios (2017)
9. da Silva, K.M.B., dos Santos, S.C.: Critical factors in agile software projects according to people, process and technology perspective. In: 2015 6th Brazilian Workshop on Agile Methods (WBMA), Pernambuco, Brazil, pp. 48–54. IEEE (2015). https://doi.org/10.1109/WBMA.2015.19
10. Wholin, C.P.R., Höst, M., Ohlsson, M., Regnell, B., Wesslön, A.: Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, New York (2000)
11. Creswell, J.W., Miller, D.L.: Determining validity in qualitative inquiry. Theory Pract. **39**, 124–130 (2000). https://doi.org/10.1207/s15430421tip3903_2