# Risk Identification and Mitigation In Agile Software Re-Engineering: A Case Study

**Chiara M. Fasching**
University of Central Lancashire

**Peggy Gregory**
University of Glasgow

**Nicholas P. Mitchell**
University of Central Lancashire

**Charlie Frowd**
University of Central Lancashire

***Keywords***: *agile, software re-engineering, legacy software, risks*

## ABSTRACT

Legacy software is becoming increasingly common, and many companies nowadays are facing the challenges associated with this phenomenon. In certain circumstances, re-engineering is the only logical way to deal with legacy software. Such projects, by their very nature, are subject to a wide variety of risks. The aim of this study was to begin building the basis of a risk framework that will support future re-engineering projects within Agile (Scrum) environments. An interpretive case study approach has been followed, where the case study was the first phase of a re-engineering process, with the method of analysis being inductive and reflexive Thematic Analysis. The dataset comprises a list of different risks that occurred during the re-engineering process. The risks observed were themed around people, processes, and technology. While technical and procedural risks are discussed in the literature, it was found that the presence of risks in social situations relating to re-engineering has been overlooked. Although these risks do not necessarily have a higher individual impact, they were found to outnumber those encountered in other aspects of the project by a significant factor. Furthermore, the social risks were often either underestimated or not even recognised. It has also been found that Scrum is an appropriate approach to re-engineering projects. Since many of the re-engineering tasks in the case study were unknown at the beginning, the flexibility brought by Scrum was an important factor in the timely and successful mitigation of emerging risks. The first contribution of this study is a comprehensive analysis of identified risks associated with one particular re-engineering project. The potential impact of those risks over a given development phase of the project, along with their actual impact, have been analysed. The second contribution discusses a proposed methodology for managing and mitigating risks in software re-engineering. It is intended that the identified risk categories form the basis of further research into different types of re-engineering projects in order to produce a more generalised framework. It is anticipated that the results presented here will help future project teams to prioritise areas of re-engineering and put adequate risk mitigation into place.

# INTRODUCTION

Legacy applications can be described as old, but well-established software systems, which are also essential for business process support (Sommerville, 2000). Legacy software does not only describe outdated technology which has accumulated technical debt, but also inherited software, which can be inflexible and with which software engineers do not know how to cope ((Bennett, 1995) and (Birchall, 2016)). Both, academics and practitioners understand that legacy applications are inflexible and expensive to maintain, although practitioners often hesitate to upgrade a system if it is not broken (Khadka et al., 2014). Besides inflexibility and the cost of maintenance, Fanelli et al. (2016) identified that "faster time to market" and "lack of experts/documentation" are the biggest drivers for practitioners to modernise legacy systems.

The source of high maintenance is often so-called technical debt, which describes numerous software quality problems. If technical debt is ignored, it may get worse (Ernst et al., 2015). It is agreed that technical debt is tightly connected to software quality (Wolff & Johann, 2015).

Sommerville (2000) states that it is necessary for companies to re-engineer legacy applications to keep them in service. The term re-engineering applies to a set of activities and techniques to tidy up the underlying structure of the application code without affecting its functionality. These activities include the analysis, redesign, restructuring, and re-implementing of the software system (Jain & Chana, 2015). The general aim of such activity is to reduce the ongoing maintenance cost of a system by improving its quality (Singh et al., 2019).

Re-engineering processes are often abandoned (Fanelli et al., 2016), which leads us to ask why, when there are so many strong drivers to modernise legacy systems. The reason for the reluctance to upgrade software can be summarised in one word: risk. Rashid et al. (2013) identify six categories of risks: user satisfaction, cost, forward engineering, reverse engineering, performance, and maintenance.

Clemons et al. (1995) suggests a framework for the identification and management of risks by supporting re-engineering as well as achieving strategic advantage by maintaining consistency between the needs of the organisation and the external environment.

Rajavat and Tokekar (2011) propose a framework for decision-driven risk engineering called ReeRisk. This theoretical framework serves to identify and eliminate risks in the early stages of the development cycle.

Another major challenge for software re-engineering is to make improvements whilst simultaneously mitigating risks and keeping the legacy application up and running. Agile, especially Scrum, is designed to deliver incremental additions while the software is in use as Scrum emphasises a working product (fully integrated and tested) at the end of every Sprint. Agile is described as a "lighter approach to building software". Instead of requirements being fixed at the beginning, in Agile the cost and the time are fixed, while the features are estimated and more flexible. This enables the software developers to prioritise the features of the application according to the business needs, which leads to on-time delivered quality software with the biggest value for the money.

This leads to the two research questions which will be answered in this paper:

- RQ1: What types of risks are encountered in a software re-engineering project?
- RQ2: How helpful are Scrum practices to support a software re-engineering process?

In this paper, we present the approach of a framework to mitigate risks in re-engineering using Scrum. Even though Agile has been described in the literature as useful for re-engineering work ((Masood & Ali, 2014) and (Holvitie et al., 2018)), there is a paucity of published research concerning Agile in re-engineering. Some grey literature sources even suggest that Agile is an inappropriate approach (Diana, 2010). Moreover, we have discovered that re-engineering risks are an uncharted area as the risks mentioned in the literature barely overlap with each other as well as our findings.

## CASE STUDY

This research was conducted through a case study exploring the first phase of the re-engineering process of a legacy application. The two goals for the first phase were to eliminate vulnerabilities and to turn it into a modern development environment while keeping it functional, both of which were achieved.

The development of the software application under consideration started 25 years ago. The system was written in non-standard C++ and was built by a single developer for a research project. Over time it was frequently expanded, not just for additional commercial functionality, but also for research purposes. Due to the absence of a planned architecture, and the many extensions, the code quality worsened over time and the code base became very messy as it included numerous redundant elements. All of this contributed to the accumulating technical debt, on top of which there was a constant need for quick bug fixes.

The re-engineering approach taken used Scrum with a team comprising five people, only one of whom was working full-time on this project (this developer was also the primary researcher). The team also included a part-time developer. The Product Owner was the researcher who had initially written the software, but who was now not in a position to do further development or modernisation on the product. The role of the Scrum Master was shared by two people from a consultant company hired to support the project.

The re-engineering work undertaken was based on a technical debt audit conducted by the consulting company. They presented their findings in two categories: software and process. The software part consisted of five different categories: coding standard, testing, build and deploy, architecture and system design, and collaboration. The re-engineering work undertaken in the category software included redundant code, breaking down and re-structuring large files, creating and automating detailed and documented user acceptance tests, updating the C++ code to conform to a language standard, and creating an installer. The modernisation of the process included migrating the system to a modern IDE, assessing external libraries as well as putting a façade on them, and implementing a Scrum process.

# TOWARDS A RISK FRAMEWORK

This study aims to build the basis of a risk framework to support future re-engineering projects within Agile (Scrum) environments. The framework will be novel as it suggests using an Agile environment for re-engineering work.

### *Research method*

The dataset comprises a list of different risks that occurred during the re-engineering process. They were collected from the Kanban board by looking for tasks undertaken to mitigate risks. Other sources were meeting notes as well as the field journal which held information about decisions, including the mitigation of risks. Also, the source code gave good insights into certain risks and how they were tackled.

Each of these risks was summarised in a single sentence. These risk descriptions were analysed using inductive Thematic Analysis (TA) (Clarke et al., 2015). This project used an inductive, and reflexive TA. This process resulted in 44 different codes. The themes, into which the sub-themes were grouped, reflect the different parts of the project. Three distinct parts could be identified, which formed the themes: ***Technology***, ***Process*** and ***People***. The ***Technology*** theme contains all the risks related to the technical side of programming and different technologies. Its sub-themes are **legacy technology**, **insufficient technology**, and **legacy application**. ***Process*** consists of all the risks which were related to managing the re-engineering work. Its sub-themes are **testing**, **time constraints**, and **lack of documentation**. ***People*** holds all the risks which are caused by human failure. Its sub-themes **are lack of knowledge**, **process engagement**, **methodology engagement**, and **social**.

Moreover, two weightings were added to each risk code to indicate a) the potential impact it could have had, and b) the actual impact it had on the project. The potential impact stated here was not assessed before the start of the project, but retrospectively, at the same time as the rest of the analysis was conducted. The potential impact was reconstructed as accurately as possible by referring to meeting notes and the Sprint board. Each weighting was assigned a logarithmic value to make the higher-impact risks stand out in the resultant graph. Risks with the weighting "none" received the value 0, with the weighting "low" the value 1, with the weighting "medium" 2, and with the weighting "high" they obtained the value 4.

### *Results*

Combining all sub-themes to display them under their themes, resulted in Figure 1 (below). This highlights the greater impact of the risk codes in the theme **People**, seen in red shadings. The impact was high throughout the project with only a little dip at the end. The initial high impact can be explained by risks related to team members lacking knowledge, which were resolved by getting to know the technologies. The methodology risks were resolved during the project as the team members got used to Scrum. The impact rose in the second half as the team started to neglect the methodology. However, more risks occurred because of an inconsistent approach to some tasks (for example: testing). Two noteworthy risks which were underestimated are "lack of team" and "lack of face-to-face working". It was expected that having such a small team would

cause stress and more work for the few team members, however, this was exacerbated by the lack of colleagues to share ideas with or discuss problems. Also, some issues would have profited from having meetings in person. Most risk codes in this theme were marked as having a low or medium impact, however, the substantial number of risk codes made it a considerable threat to the success of the project. Moreover, most of the risk codes remained a constant threat throughout the first phase.

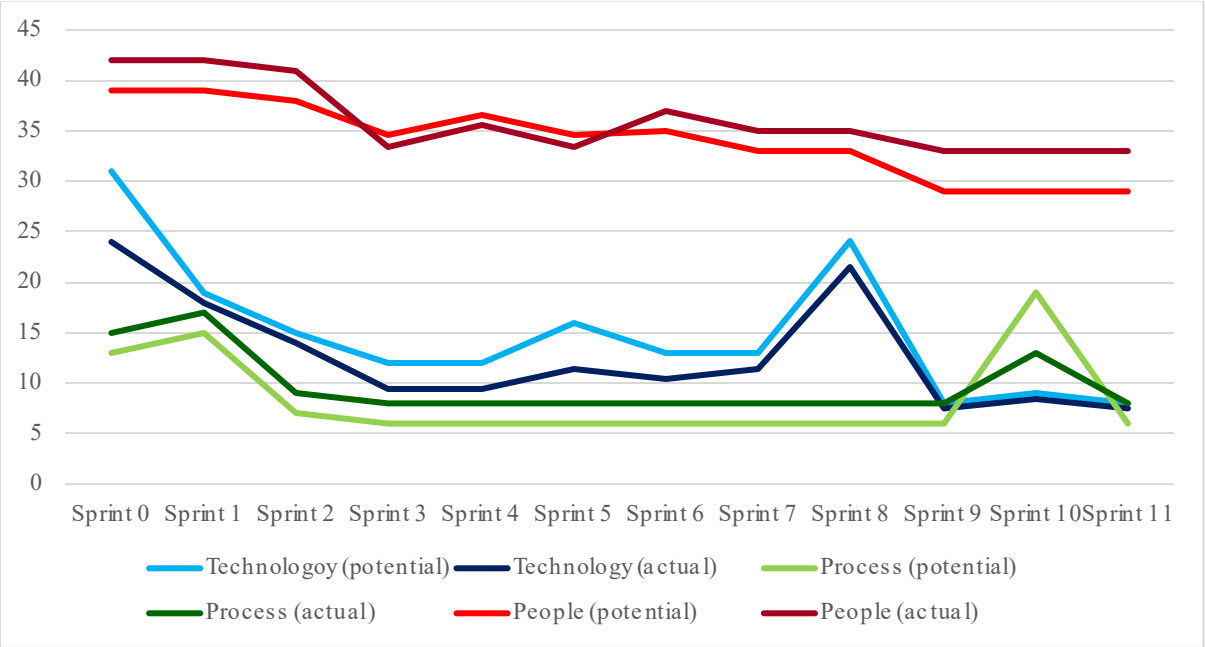


*Figure 1: Weighted risk codes frequency within a theme per Sprint*

Another striking detail in this diagram is that the theme **Technology** had a high expected impact in the beginning as there was a risk that the software would not work on a newer OS, so it needed to be migrated to a new IDE. This risk included the difficulty of integrating the code and the possibility it might not work as intended because changes were necessary to meet new language standards. However, the impact of these risks did not transpire. Moreover, the risk impact of this theme peaked in Sprint 8. This was due to the difficulty and eventual failure of integrating specific new technology into the legacy system. The fact that less coding was undertaken than expected meant that some risks had no impact at all.

The lack of documentation posed a high-impact risk at the beginning of the theme **Process**. This risk was resolved after documentation was added. It is also notable that the impact suddenly increased in Sprint 10 due to not having the application tested thoroughly at the end of Phase 1. Moreover, due to running out of time at the end of Phase 1, some tasks could not be executed and therefore risk codes related to having a lack of time emerged.

## AGILE IN RE-ENGINEERING

As Scrum is a part of Agile, Scrum follows Agile practices as well as its guidelines. As mentioned, Scrum was chosen for this re-engineering project because its flexibility is a good response to the unpredictability of challenges and risks.

Table 1 highlights the risks, which were mitigated by different Scrum techniques.

| Risks | Mitigation strategies using Scrum |
|---|---|
| The precise nature of re-engineering activities was not known in advance. | Continual re-prioritisation of tasks, and planning in detail for the short term (Sprint planning / Backlog grooming) |
| Previously unidentified risks became apparent as the project progressed. | Revising and adapting ways of working within the team to address risks as they arise (Sprint Review) |
| The causes of the struggles were difficult to identify. | Discussing issues during Sprint Retrospectives helped to uncover the actual risk that caused them |
| Being stuck | Highlighting potential sticking points early (Daily Stand-up) so that technical mitigations can be put in place (also feeding into Sprint Review and Planning) |
| Lack of knowledge | Highlighting the issue during Daily Stand-ups to receive help from team members (also feeding into Sprint Review and Planning). |
| Having struggled with time management because of working part-time on the project | Discussing completed work and planning work for the next few days during Daily Stand-ups |
| Difficulties explaining to stakeholders what was done in previous Sprints and how to showcase it | Adding a Definition of Done to every user story and collecting evidence for every completed task to document progress. |

*Table 1: Summary of risks mitigated by using Scrum practices*

## LIMITATIONS

This study is based on a single case study, it is therefore difficult to generalise the risks found here to those that might occur in other re-engineering projects. Also, few conclusions about the probability of the risks can be made without observing multiple re-engineering processes. Some risk categories discussed in the literature are absent from this study, because it was based on one phase of a re-engineering project. Furthermore, the presentation of findings in Figure 1 does not show when each risk became apparent.

Another major drawback of this study is that the potential impact was not assessed at the beginning of the project but retrospectively. This made it impossible to evaluate how well the team estimated risks. Finally, there was a strong subjective element to this work as the primary researcher was directly involved in this project as a software engineer, which could be considered as a limitation. The weightings for each impact were subjective as they were determined solely by the primary researcher.

## DISCUSSION AND CONCLUSION

To answer the first research question, a set of risks including their impact on a re-engineering project has been produced. This makes it interesting to look at the risk categories of previous

papers and compare them with my results. Risks mentioned in previous papers (for example: Rashid et al. (2013) and Clemons et al. (1995)) barely overlapped with mine.

Social risks seem to be ignored in some papers, e.g.: (Rashid et al., 2013). Other papers mention certain team or social risks, such as (Khadka et al., 2014). They describe the reluctance of software developers to modernise legacy applications as they often conceive them as their "baby", or they fear redundancy following the modernisation process, so they refuse to share their knowledge. Further social risks addressed in the paper are the non-understanding of managers for the need for modernisation, and the reluctance of providing a sufficient budget for it. However, they do not mention any risks which could occur within the team or even related to a single person. This proves that social risks are often overlooked or forgotten about. Even though they might not be directly related to the project - in the form of the actual software development work – and do not seem to be obvious, they are as critical, or even more, than other risks. The novelty of this study stems from the fact that social risks were taken into consideration, as they were found to have a major impact, and are not just being mentioned as a side issue.

It may be the case that some risks which appeared at specific junctures in this project could be persistent threats in others, such as *inconsistent testing*. Also, some risks are not time-bound, such as the risk code *fix is worse than problem*. Comparing the risk categories from Rashid et al. (2013) and Clemons et al. (1995) to mine, the reasons for the differences can be justified. Financial risks did not appear in my analysis as the budget for the project had already been approved when the contribution of the primary researcher started. Forward and reverse engineering, maintenance, and performance risks did not appear as the re-engineering process was not advanced enough in the First Phase. As the original system designer was part of the re-engineering team, functionality risks, such as the system not meeting present or future needs, were not perceived as a threat. Finally, political conflicts did not endanger the success of the project as the re-engineering work was of high importance to the organisation.

Addressing the second research question, although Scrum has been traditionally viewed as a mechanism to manage and prioritise the implementation of *new* functionality, it was found that by treating the mitigation of technical debts as functional requirements, Scrum could be equally well applied. Alongside this, the management and mitigation of the uncovered risks, especially social risks, made Scrum an appropriate approach for re-engineering legacy software. Suggesting an Agile environment for re-engineering work is the second novel feature of this study.

Although Agile has been mentioned in the context of re-engineering before, it has not to our knowledge been suggested as the preferred approach. Holvitie et al. (2018) surveyed practitioners and found that Agile practices are perceived to have a generally positive effect on managing technical debts. Some of their findings overlap with mine e.g.: most practitioners perceived iteration reviews/retrospectives and adhering to coding standards as having a positive effect on managing technical debt. However, practices I viewed as useful, such as on-site customers, were mostly perceived as neutral, and core practices of Agile, like iterations, backlogs, and daily meetings, were rated as having a positive impact by only 50-60% of the participants. The differences between my results and Holvitie et al.'s may emerge from the fact

that their participants were software engineers who did not specifically work on legacy applications. Only 40% of the participants had good knowledge about technical debt.

Singh et al. (2019) proposed a framework, which is supported by a case study, using Agile methodology as the flexible methodology fits nicely to the requirements of a re-engineering process. A serious limitation of this work is that the authors only tested their framework over a single Sprint while reducing a set of code complexity.

The set of risks, alongside an Agile approach to their management, has the potential to form the basis of further research into different types of re-engineering projects, leading further towards a more generalised framework.

## REFERENCES


Bennett, K. (1995). Legacy Systems: Coping with success [Article]. *IEEE Software*, *12*(1), 19-23. https://doi.org/10.1109/52.363157

Birchall, C. (2016). *Re-engineering legacy software*. Simon and Schuster.

Clarke, V., Braun, V., & Hayfield, N. (2015). Thematic analysis. *Qualitative psychology: A practical guide to research methods*, *222*, 248.

Clemons, E. K., Row, M. C., & Thatcher, M. E. (1995). An integrative framework for identifying and managing risks associated with large scale reengineering efforts. Proceedings of the Annual Hawaii International Conference on System Sciences,

Diana, R. (2010). Re-Engineering In Agile Development Can Just Be Refactoring. In *Agile Zone*.

Ernst, N. A., Bellomo, S., Ozkaya, I., Nord, R. L., & Gorton, I. (2015). *Measure it? Manage it? Ignore it? software practitioners and technical debt* Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy. https://doi.org/10.1145/2786805.2786848

Fanelli, T. C., Simons, S. C., & Banerjee, S. (2016). A systematic framework for modernizing legacy application systems. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016,

Holvitie, J., Licorish, S. A., Spínola, R. O., Hyrynsalmi, S., MacDonell, S. G., Mendes, T. S., Buchan, J., & Leppänen, V. (2018). Technical debt and agile software development practices and processes: An industry practitioner survey [Article]. *Information and Software Technology*, *96*, 141-160. https://doi.org/10.1016/j.infsof.2017.11.015

Jain, S., & Chana, I. (2015). *Modernization of Legacy Systems: A Generalised Roadmap* Proceedings of the Sixth International Conference on Computer and Communication Technology 2015, Allahabad, India. https://doi.org/10.1145/2818567.2818579

Khadka, R., Batlajery, B. V., Saeidi, A. M., Jansen, S., & Hage, J. (2014). *How do professionals perceive legacy systems and software modernization?* Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India. https://doi.org/10.1145/2568225.2568318

Masood, A., & Ali, M. A. (2014). Applying Agile Requirements Engineering Approach for Re-engineering & Changes in existing Brownfield Adaptive Systems. *arXiv preprint arXiv:1410.6902*.

Rajavat, A., & Tokekar, V. (2011). ReeRisk–A Decisional Risk Engineering Framework for Legacy System Rejuvenation through Reengineering. Computer Networks and Information Technologies: Second International Conference on Advances in

Communication, Network, and Computing, CNC 2011, Bangalore, India, March 10-11, 2011. Proceedings 2,

Rashid, N., Salam, M., Sani, R. K. S., & Alam, F. (2013). Analysis of risks in re-engineering software systems. *International Journal of Computer Applications*, *73*(11), 5-8.

Singh, J., Singh, K., & Singh, J. (2019). Reengineering framework to enhance the performance of existing software [Article]. *International Journal of Advanced Computer Science and Applications*, *10*(5), 536-543. https://doi.org/10.14569/ijacsa.2019.0100570

Sommerville, I. (2000). 28. Software Re-engineering.

Wolff, E., & Johann, S. (2015). Technical Debt. *32*(04), 94-c93. https://doi.org/10.1109/ms.2015.95