# Reallocation of Resources during Releases for Better Outcome in Software Development

£Md. Tajmilur Rahman, £Peter C. Rigby

£Department of Engineering and Computer Science, Concordia University, Montreal Quebec, Canada

# Abstract

...

# I. Introduction

Software projects are notorious for going over budget and schedule. Rush periods are often gets seen before a major release that turns the developers into dinosaurs as Frederick Brooks likens in his benchmark study “*The Mythical Man Month*” [1]. This “Rush To Release (RTR)” can be prompted either by external forces such as decision by management to include new features in the release or to release earlier to beat a competitor. Alternatively, the rush may simply be due to inappropriate or unrealistic scheduling. Whatever the reason is it is an obvious. Regardless of the causes, the rush to release stresses developers and often requires developers to work on unusual, high priority or critical areas of the system. In this paper we study how RTR effects project organization and introduces technical debt. The key research questions that we expect to answer with our methodology are as follows:

*1) Do developer work on different areas of the system around the time of release?*

*2) Are there certain areas of the system that receive increased attention (i. e. do developers focus on a smaller set of files around releases).*

*3) Do the areas of code that are modified around the time of release have higher defect densities than code that is modified during normal development?*

We observe a reallocation of the resources among the software development teams in a large project to identify that an improper reallocation or inappropriate reorganization causes a disruptive event take place in a software development process. We attempt to identify a project’s different release times and calculate the difference between two consecutive releases to discover which are the new areas been worked in between releases. The commit log data that we are working on for this purpose will help us to extract a lot of information like calculating the developers' working areas and time-frame of each release. These information will help us to identify the criteria of the resources, their role, file ownership and nativeness in the domain. This knowledge of nativeness will guide us understanding the way of reallocation.

Very few research works have been performed regarding the re-allocation of resources. Robert van Engelen worked for similar kind of a research to understand the resource allocation dynamics across the software projects [2]. He mainly tried to reallocate development resources amongst projects for increasing the satisfactory level of consumer or customer while we are focusing on the impact on code-base like the complexity of script files. Robert proposed a project entropy metric in his work to understand if there is any limit for a particular reallocation does not lead to user satisfaction. Here entropy is to represent disorder and chaos to understand degradation of software and its inherent complexity. In his work resources may not just be the developers but also can be any other resources necessary for a software project development.

We have organized this paper as follows. In Section II, we describe some background and motivations followed by some summaries of related works in section III. Section IV will describe about the ownership of files and ownership of a set of files or a directories. We will try to understand how native a code-base is to a developer or a development team. In Section V, some analysis to determine reallocation has been performed in a release or where reallocation needs to be performed will be presented. What changes in nativeness (∆ƞ) occurs after the reallocation. Section VI will give us the result to show how change in ∆ƞ puts impact on the outcome of a software. Finally section VII will give us an idea of our future work followed by the a conclusion in sectin VIII.

# II. Background and Motivation

Since the invention of the idea of softwares engineering discipline has proposed number of models for the development process of a software. Linear approach like Water-Fall model [3], iterative approach like Spiral model [4] and there are many others, but over the years we also have observed that there is no software model that can be implemented 100% in a software development industry in practical. There are many practical issues take part in the life cycle of software development. Some times project members with interdependent tasks usually may not communicate effectively, coordination breakdowns occur which results in integration failures [5]. There may have lower developers productivity [6][7] which may cause inefficient run in the rush moments in a release period. There is a substantial and important body of literature on risk in software engineering. Boehm identified the most important risks encountered by software project managers and described successful risk management practices [8, 9, 10]. Some of the risks identified are related to disruptive events, such as the introduction of a new technology, but most are macro risks associated with running a project, such as developing the wrong functionality. General risk mitigation strategies can be difficult to apply to specific disruptive events. There may be various kinds of disruptive events for example as a release approaches, developers take shortcuts that introduce technical debt. If it is not repaired, the longterm quality of the system will suffer. Another example, if a lead developer who owns an important part of the code-base leaves and if steps to train other developers were not taken, it will become a dead area of the system and will be difficult to modify and maintain. Also often management reorganizes the developers on a company’s projects, with the result that developers move to code-bases for which they have less experience. The reorganization introduces new perspectives and expertise that can lead to innovation; however, it can also result in a drop in productivity and the unnecessary re-writing of large portions of the system that the new developers do not understand.

In this paper, we plan to take the measures on this last example among them mentioned above. We want to study that proper re-organization or efficient re-allocation of resources based on meaningful criteria can bring better outcome of a software development project by identifying quantify outcomes (Example: number of defects found).

# III. Related Works

Many people has worked with pretty relevant ideas but I didn't find many work very similar to our motivation. Abram Hindle and Michael W. Godfrey worked on release pattern discovery via partitioning [11]. In this research they proposed a method of observing, analyzing and summarizing the results of metrics of revisions found near releases. They have characterized a project's behavior around the time of major and minor releases. This is done by partitioning the observed activities like the art-effect check-ins around the dates of major and minor releases, then look for reasonable patterns. Hindle divided the revisions in each release in 4 different classes, Source Code, Testing, Building, and Documentations. Actually this paper worked in a reverse way than Cook did. Cook inserted sensors and monitors into the development process but Hindle and Michael analyzed the data to understand what happened in the past.

Another research work we would like to mention was done by Daniela Damian and Remko Helms with some others where they have worked on the role of domain knowledge and cross functional communication among the OOS development teams [12].

# IV. Ownership and Native-Code

…

# V. Research and Analysis

…

# VI. Results and Discussion

…

# VII. Future Work

…

# VII. Conclusion

…

# References

1. F.B. Brooks, J. The MythicalMan Month, Addison-Wesley, Reading, Mass., 1975.
2. Robert van Engelen, Subhajit Datta *“Project-entropy: A Metric to Understand Resource Allocation Dynamics across Software Projects”.*
3. W. W. Royce. *“Managing the development of large software systems: concepts and techniques”*. In Proceedings of the 9th International Conference on Software Engineering, pages 328–339. IEEE Computer Society Press, Mar. 1987.
4. B. Boehm. *“A spiral model of software development and enhancement”*. SIGSOFT Softw. Eng. Notes, 11(4):14–24, 1986.
5. I. Kwan, A. Schroter, and D. Damian, *“Does socio-technical congruence have an effect on software build success? A study of coordination in a software project”*, IEEE Transactions on Software Engineering, 37(3), May-June 2011, pp. 307-324.
6. M. Cataldo, P. Wagstrom, J. Herbsleb and K. Carley, *“Identification of coordination requirements: implications for the design of collaboration and awareness tools”* in CSCW, 2006, pp. 353–362.
7. D. Damian, L. Izquierdo, J. Singer and I. Kwan, *“Awareness in the Wild: Why Communication Breakdowns Occur”*, In International. Conference on Global Software Engineering, 2007, pp. 81-90.
8. B. W. Boehm. Software risk management: Principles and practices. IEEE Softw., 8(1):32–41, January 1991.
9. M. Keil, P. E. Cule, K. Lyytinen, and R. C. Schmidt. A framework for identifying software project risks. Commun. ACM, 41(11):76–83, Nov. 1998.
10. B. Boehm. Get ready for agile methods, with care. ACM Computer, 35(1):64 –69, 2002.
11. Hindle, A., Godfrey, M.W., Holt, R.C. Fourth International Workshop on Mining Software Repositories. ICSE Workshops MSR, 20-26 May 2007, Minneapolis, MN.
12. Damian Daniela, Helms Remko, Kwan Irwin, Marczak Sabrina, Koelewijn Benjamin *“The role of domain knowledge and cross-functional communication in socio-technical coordination”* 35th International Conference on Software Engineering (ICSE), 2013, Page(s): 442 - 451.