# Re-Allocation of Resources during Releases for Better Outcome in Software Development [*]

Md Tajmilur Rahman
Concordia University
Montreal, QC H3G 1M8
438-932-2288, +1
rupak.karmadhar@gmail.com

Peter C. Rigby
Concordia University
Montreal, QC H3G 1M8
514-848-2424, +1
peter.rigby@concordia.ca

## ABSTRACT
This section will be written at the end.

## Categories and Subject Descriptors
K.6.3 [**Software Management**]: Software Development; Software Resource Management, Resource Reallocation

## General Terms
Experimentation, Human Factors, Resource Management, Reallocation.

## Keywords
Resource Reallocation, Software Releases

## 1. 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 decisions 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 developers 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 projectâĂŹs different release times and calculate the difference between two consecutive releases to discover which are the new areas have been worked between two consecutive 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. This information will help us to identify the criteria of the resources, their roles, file ownership and nativeness (âĹĘÆđ) in the domain. This knowledge of nativeness (âĹĘÆđ) will guide us understanding the method 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 [4]. 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 2, we describe some background and motivations followed by some summaries of related works in section 3. Section 4 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 5, 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 (âĹĘÆđ) occurs after the reallocation. Section 6 will give us the result to show how change in âĹĘÆđ puts impact on

---

the outcome of a software. Finally section 7 will give us an idea of our future work followed by the conclusion in section 8.

## 2. BACKGROUND

Sometimes 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 [2, 3] 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 [6, 7, 8]. 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 long term quality of the system will suffer. Another example can be placed, 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 companyâĂŹs 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).

## 3. RELATED WORKS

Many people have worked with pretty relevant ideas but I didn't find many very similar to our motivation. Hindle worked on release pattern discovery via partitioning [9]. 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 checkins 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 [10]. 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 Damian where they have worked on the role of domain knowledge and cross functional communication among the OOS development teams [11]. Posnett did some dual ecological measures of focus in software development [12]. Posnett's measure was for the more general view that unifies developer focus and artifact ownership. Posnett analyzed i) developer artifact contribution to network to a predator-prey food web ii) drew upon ideas from echology to produce a novel and iii) conceptually unified view of measuring focus and ownership. Another study was done by F. Rahman about the authorship of the code-bases in OSS development [13].

## 4. METHODOLOGY AND DATA

This section presents our methodology for discovering information which can give us the idea to get the answers to our research questions. We have collected the development history data of Linux kernel. Actually it is a database containing the all the commit log records by the Linux kernel developers since 2005. We are going to present the steps involved in this process and then we will follow up with an application of our methodology in a case study. Our methodology can be summarized as: Extracting Data for revisions and releases (Section 4.1); Partitioning the version numbers (Section 4.2); Get time-span between each release (Section 4.3); Calculate developer areas (Section 4.4); Finding code area owners (Section 4.5). Finding merging time and development time within a release period that we found in section 4.1 (Section 4.6).

### 4.1 Extracting Data
... ... ...

### 4.2 Partioning Version Numbers
... ... ...

#### 4.2.1 Inline (In-text) Equations
... ... ... ... ... ...

### 4.3 Get Time-Span Between Release
... ... ...

### 4.4 Calculate Developers' Area
... ... ...

### 4.5 Finding Code-Area Owners
... ... ...

### 4.6 Extracting Data
... ... ...

#### 4.6.1 Display Equations
... ... ... ... ... ...

## 5. CONCLUSIONS

This section will be written at the end.

## 6. ACKNOWLEDGMENTS
... ... ...

## 7. REFERENCES

[1] F. Brooks. *The Mythical Man-Month*. Addison-Wesley, 1975.

[2] M. Cataldo. Identification of coordination requirements: implications for the design of collaboration and awareness tools. *CSCW*, pages 353–362, May-June 2006.

[3] D. Damian. Awareness in the wild: Why communication breakdowns occur. pages 81–90, 2007.

[4] S. Datta. Project-entropy: A metric to understand resource allocation dynamics across software projects.

[5] I. Kwan. 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):307–324, May-June 2011.

# APPENDIX
# A. HEADINGS IN APPENDICES

Apendix section will be written later with the following subsections may be:

## A.1 Introduction

...

## A.2 Background and Motivation

### A.2.1 Related Works

...

### A.2.2 Methodology and Data

...

## A.3 Conclusions

...

## A.4 Acknowledgments

...

## A.5 References

... ...