PREDICTING PROBLEMS IN A LARGE SOFTWARE PROJECT THROUGH ITS BUILD SYSTEM

A THESIS PROPOSAL SUBMITTED TO MY THESIS COMMITTEE

MASTER OF SCIENCE

IN

INFORMATION AND COMPUTER SCIENCES

By
Aaron A. Kagawa

Thesis Committee:

Philip M. Johnson, Chairperson

November 19, 2003
Version 1.1.0

Abstract

Managing large software projects is intrinsically difficult. Although, high software quailty is a definite must, other issues like time and cost play major roles in large software development. For example, if a software company can produce the highest quality products but cannot predict how long and how much it is going to cost, then that company will not have any business.

Software metrics are one answer to those problems. Software metrics are the measurement of periodic progress towards a goal [3]. Metrics are used to indicate various problems in a development process.

Currently, there are a large number of documented metrics. However, there does not exist one perfect formula to to satisfy every development's quality and productivity. The key to a good software metrics program is to be able to identify the specific goals of the development and be able to assist in reaching these goals.

I will addres this concept through the development of specific measurements and analyses that will improve the quality of a specific system, the Mission Data System (MDS) at the Jet Propulsion Laboratory. I will attempt to identify certain software metrics that can help JPL reach their development goals.

To accomplish this I have created the Hackystat Jet Propulstion Laboratory Build System (hackyJPLBuild). This system measures and analyzes the build system of MDS. The research question of this thesis is, is it possible to collect data from a build process of a large scale software project, in order to understand, predict, and prevent problems in the quality and productivity of the actual system. To evaluate this research question I will conduct three case studies: (1) can the hackyJPLBuild system accurately represent the build process of MDS, (2) can threshold values indicate problematic issues in MDS, and (3) can hackyJPLBuild predict future problematic issues in MDS.

Initial results of case study 1 indicate that hackyJPLBuild can accurately represent the build process of MDS. In fact, hackyJPLBuild has already identified some pontetial flaws in the MDS build process. Case studies 2 and 3 have not been conducted yet.

If this thesis project is successful, I will be able to identify a generic set of build process software metrics that are applicable to identify, predict and prevent software qualilty and productivity problems.

Table of Contents

Chapter 1

Introduction

Managing large software projects is intrinsically difficult. Although, high software projects is a definite must, other issues like time and cost plays major roles in large software development. “The fundamental issue is an economical imperative: Produce function at a cost, quality, and schedule that meet users’ needs [1].” For example, if a software company can produce the highest quality products but cannot predict how long and how much it is going to cost, that company will not have any business.

There are many ways to measure software development in order to assess quality and productivity. Traditionally, managers asess quality through measures like defect density, and productivity is assessed through measures like lines of code per hour. The thesis of this research is that, for a large scale programming projects is it possible to predict quality and productivity by measuring and monitoring the system’s build process. If it is possible to prodict problems with a large software proejct, then is it possible to prevent these problems from ever occuring. I will address these research questions through the development of specific measurements and analyses of a specific large scale programming project: the Mission Data System (MDS) at the Jet Propulsion Laboratory. If sucessful, my project will not only provide useful support to the software development of MDS, but it will also lead to more general hypotheses. First, I will be able to make hypotheses concerning the kinds of build measurements and analyses that are useful for predicting quality and producitivity. Second, the kinds of large scale programming projects that would be amemble to this research. And last of all, the kinds of improvements in quality and productivity that can result.

1.1 The Problem of Large Software Systems

“The scale of large systems has grown by three orders of magnitude in the last thirty years, and this rate of growth is likely to continue or even increase in the future [1].” Software is growing in

size and maybe even complexity, yet the understanding of large software systems are relatively low. NEED MORE GENERAL INFORMATION ABOUT THE PROBLEMS OF LARGE SOFTWARE SYSTEMS.

1.2 The Mission Data System

The Mission Data System at the Jet Propulsion Laboratory is a project to provide all space mission software with a common architecture. It also provides autonomous tracking of the state of the spacecraft and the system uses that information to make its own autonomous decisions. [4]

1.3 The Hackystat Jet Propulsion Laboratory Build System: A system to predict and prevent problems in MDS.

The Hackystat Jet Propulsion Laboratory Build System (hackyJPLBuild) is a specialized extension of the Hackystat System. Hackystat is a framework for unobtrusive metric collection and analysis. To use Hackystat, developers download and install "sensors" in development tools. These sensors collect data of the development that occurs, send this data to a server, which analyzes that data to provide insights back to the developers to support improvements in their work products and process.

The hackyJPLBuild system takes alters the traditional use of the "sensor" and attaches a sensor to the MDS CCC Harvest build tool. The MDS CCC Harvest build tool sensor will collect data pertaining to the build process of MDS. The hackyJPLBuild system uses this data to analyze certain aspects of the build process.

MORE DETAILS ABOUT HACKYSTAT AND hackyJPLBuild to come.

1.4 Thesis Statement

This research investigates the effectiveness of hackyJPLBuild system and gathers qualitative and quanititative data in order to assess the following general hypothesis:

Thesis Statement:

1. hackyJPLBuild can accurately represent the software process of the Mission Data System.

2. hackyJPLBuild will identify threshold values that indicate above average attributes in the Mission Data System.
3. hackyJPLBuild will predict software problems in the Mission Data System.
4. hackyJPLBuild will help improve developers' and managers' understanding of the MDS system and the systems build process.

The first hypothesis claims that the hackyJPLBuild system along with the Hackystat System has the ability to accurately represent the software process of the Mission Data System. To evaluate this claim, I will conduct Case Study 1 - Secion 4.1, which validates several implementation level details of the hackyJPLBuild System.

The second hypothesis claims that the hackyJPLBuild system will be able to identify threshold values that indicate "above average" attributes with in the software process of the Mission Data System. An example of a threshold is the age measurement, which measures the time it takes for a MDS Package to reach the Release Harvest State. To evaluate if hackyJPLBuild can recognize these thresholds, I will conduct Case Study 2 - Section 4.2, in which I will attempt to create specialized Hackystat analyses to identify threshold values for the Age and Throughput measurements.

The third hypothesis claims that the hackyJPLBuild system will be able to predict software process problems using the threshold values generated in claim 2. To evaluate this claim I will conduct Case Study 3 - Section 4.3, in which I will use threshold values to predict future software process problems.

The last hypothesis claims that the hackyJPLBuild system will increase the improve the understanding that the developers and managers have of the Mission Data System and the system's build process. To evaluate this claim, I will conduct several qualitative surveys and telephone conferences to with the developers and managers of the MDS system.

1.5 Structure of the Proposal

The remainder of this thesis proposal is as follows. Chapter 2 discusses previous studies that influenced this research. Chapter 3 describes the functionality and architecture of the hackyJPLBuild system. Chapter 4 discusses the experimental design. Chapter 5 discusses the expected results and sample analyses. Finally, Chapter 6 contains my thesis plan.

Chapter 3

hackyJPLBuild System Architecture and Design

This chapter provides a detailed explanation of the Hackystat Jet Propulsion Laboratory Build System (hackyJPLBuild). The initial requirements of this system developed by Dr. Philip Johnson [2]. Using this proposal, I designed and implemented this system. The following sections provide explanations of the Sensor Data Types, Abstract Data Types, and Analyses that the hackyJPLBuild system provides.

3.1 Sensor Data Types

The hackyJPLBuild system provides two Sensor Data Types; the StateChange and Build Sensor Data Types. To populate the data for these SDTs a specially designed Hackystat sensor will be attached to the MDS Harvest CM Tool.

3.1.1 State Change Sensor Data Type

The StateChange Sensor Data Type represents *promotion* or *demotion* events of a MDS Package and aspects about the pacakge's state at this point in time. It contains the following fields:

State Change Sensor Data Type Attributes:

1. *tstamp*: A UTC timestamp (i.e. the number of milliseconds since 9/1/1970. Example: "105071895265" (which is roughly 4/18/2003 2:08pm). Every StateChange instance must have a unique tstamp, and this tstamp should represent the time at which the promotion or demotion took place.

2. *packageId*: The unique identifier for the MDS Package that is being promoted or demoted. Example: “CP-00971”.
3. *startState*: The Harvest CM State this MDS Package was in just before the promotion/demotion. Example: “Dev”.
4. *endState*: The Harvest CM State this MDS Package is now in as a result of the promotion/demotion. Example: “Dev-Complete”.
5. *newFiles*: A comma delimited list of file names that have been added since the last state change.
6. *modifiedFiles*: A comma delimited list of file names that have been changed since the last state change.
7. *deletedFiles*: A comma delimited list of file names that have been deleted since the last state change.
8. *unchangedFiles*: A comma delimited list of file names that have not been changed since the last state change.
9. *iar*: If this MDS Package (such as a CP) associated with an IAR, the IAR packageId is provided here. Example: “IAR-02068”.
10. *developer*: The username of the developer responsible for this MDS Package. Example: “cxing”.

The following is an example of a StateChange Sensor Data Entry in XML form.

```
<?xml version=''1.0'' encoding=''UTF-8'' ?>
<sensor>
  <entry tstamp=''97847280003'' tool=''harvest'' pacakgeId=''CP-00971''
    startState=''dev'' endState=''dev_complete'' newFiles=''newFile.cc''
    modifiedFiles=''modifiedFile.cc'' deletedFiles=''deletedFile.cc''
    unchangedFiles=''unchangedFile.cc'' iar=''IAR-02068''
    developer=''cxing'' />
</sensor>
```

3.1.2 Build Sensor Data Type

The Build Sensor Data Type represents an occurence of the build with a given MDS Package. It contains the following fields:

Build Sensor Data Type Attributes:

1. *tstamp*: A UTC timestamp representing the time at which the build took place. Example: "1050710895265".
2. *packageId*: The unique identifier for the MDS Package that is being built. Example: "CP-00971".
3. *testPassed*: A number indicating the total passed tests. Example: "25".
4. *testFailed*: A number indicating the total failed tests. Example: "0".
5. *failureType*: One of none, compile, link, and run-time.

The following is an example of a Build Sensor Data Entry in XML form.

```
<?xml version=''1.0'' encoding=''UTF-8'' ?>
<sensor>
  <entry tstamp=''97847280003'' tool=''harvest'' pacakgeId=''CP-00971''
    testsPassed=''10'' testsFailed=''0'' failureType=''none'' />
</sensor>
```

3.1.3 Sending data to the server

Sensor Data Types define the type of data that can be sent to a Hackystat server. To accomplish sending data, a Hackystat Sensor must be attached to the development tool which we wish to monitor, in this case, Harvest. This sensor has been partly designed by Rich Hug of the Jet Propulsion Laboratory, the local Harvest expert. He developed a script which extracts data from Harvest into a text file. This text file can then be used as input for the Harvest Sensor.

We have found that sending all the data from January 2003 to October 2003 to a Hackystat server has a duration of a few seconds.

3.1.4 Data in the server

Once the Sensor Data is in a Hackystat server, the Sensor Data is used as the primary source of data, refered to as “Raw Data”. This raw data is the finest grained source of data the hackyJPLBuild analyzes. However, analyzing raw data is computationallyn expensive becuase it is so fine grained. The following section discusses how I solved this problem.

3.2 MDS, Work, and Rollup Packages

The Sensor Data Types provide access to fine grained measurements of the Harvest Tool. However, the raw data is computationally expensive. Therefore, a larger grained measurement was needed. The goal of creating a larger grained measurement is to provide aggregated, easily accessable, and derived data that cannot be obtained from just the raw data. To represent this larger grained measurement the hackyJPLBuild system collects together all Sensor Data Entries related to a specific MDS Package into an Abstract Data Type (ADT).

There are three different types of ADTs; (1) the MDS Package, (2) the Work Package, and (3) the Rollup Package. Figure 1 provides a UML diagram of the ADTs.

The following explains the attributes of each ADT.

MDS Package:

1. *packageID*: The unique identifier of a MDS Package
2. *startState*: The first Harvest CM State of the MDS Package. This is determined by getting the startState of the first StateChange Sensor Data Entry associated with this MDS Package.
3. *endState*: The last Harvest CM State of the MDS Package. This is determined by getting the endState of the last StateChange Sensor Data Entry associated with this MDS Package.
4. *developers*: The developers that have contributed to this MDS Package.

Rollup Package:

1. *type*: One of ITR, IT, or CPR.

Work Package:

1. *type*: One of CP, IAR, IM, or RP.

2. *newFiles*: The total number of new files in this Work Package.
3. *modifiedFiles*: The total number of modified files in this Work Package.
4. *deletedFiles*: The total number of deleted files in this Work Package.
5. *iars*: The IARs associated with this Work Package.

There are several things to note about the representations provided here. (1) A MDS Package is either a Rollup Package or a Work Package. (2) Rollup Packages represent documentation level Pacakges. (3) Work Packages represent where the actual "work" is done. For this reason, Work Package are the only Package that contains attributes for files, because "work" cannot be done on Rollup Packages.

3.3 hackyJPLBuild Analyses

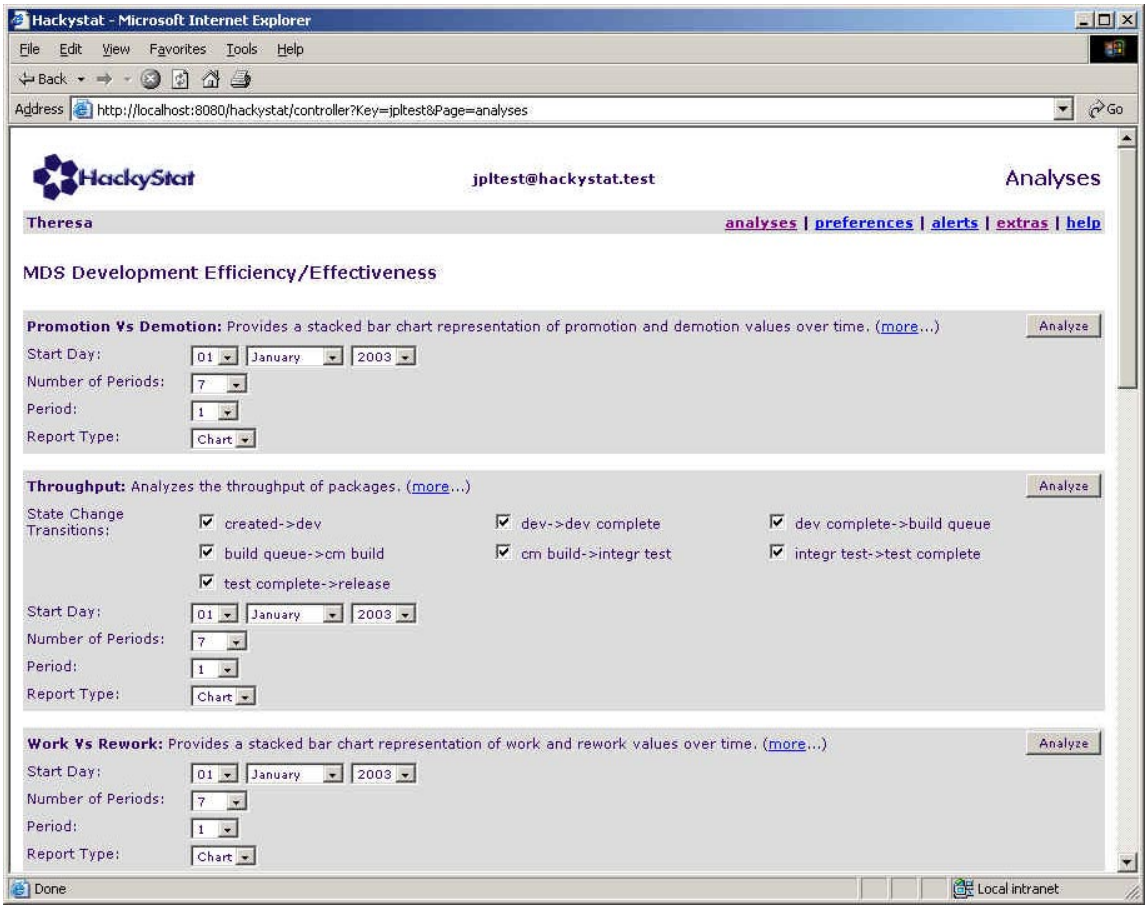

Figure 3.1. hackyJPLBuild MDS Development Efficiency Analysis Set

Chapter 4

Experimental Design

The experimental design of this research evaluates my thesis claims explained in Section 1.4. I propose to accomplish this by conducting three case struides in successive order. First, I will conduct a case study to understand and validate the hackyJPLBuild representation of the MDS CCC Harvest build process. Second, I will conduct a statistical analysis of historical data in hopes of identifying factors that influence problems in the build system. Third, I will conduct a case study to determine if the hackyJPLBuild sysem can predict problems in the build system.

4.1 Case Study 1 - Understanding and Validating

For the hackyJPLBuild system to be successful, it must be able to accurately represent the build process of the MDS CCC Harvest Tool. Case Study 1 includes understanding and validating data from the MDS CCC Harvest Tool.

Prior to August 2003, Dr. Philip Johnson and I created the hackyJPLBuild system purely on assumptions we had on the data we would receive from the MDS CCC Harvest data. In August 2003, I recieved the shell command file for both the Build Sensor Data and State Change Sensor Data. These files represent MDS work from January 2003 to August 2003 (which I will call YTD, Year-To-Data, from this point on). The YTD data was then loaded into the hackyJPLBuild system.

In this case study, I have conducted several mini studies on the data (1) to validate my understanding of the data, (2) validate how the hackyJPLBuild system represents this data, and (3) to validate the correctness of the data. I have provided the initial findings of these mini-studies in the following sections. Dr. Philip Johnson and I have gathered more detailed explanation of the findings for a White Paper which was presented for review by the MDS managers.

It should be noted that these mini studies are not based on any specific hypotheses; rather it is a continual goal to further enhance my understanding of the MDS CCC Harvest Tool and MDS,

enhance how hackyJPLBuild represents and analyzes the data, and to validate the actual data in the hackyJPLBuild system. Therefore, when interesting opportunities arise to explore the data, I will analyze them under this case study.

4.1.1 Understanding the MDS Architectural Elements

MDS Architectural Elements are defined as the incremental development element. For example, there are Implementation Task Rollup, Implementation Tasks, Change Package Rollups, Verification Rollups, Requirement Package, Change Package, Internal Anomaly Report, etc. The goal of this mini-study is to determine how the hackyJPLBuild system can represent these elements in the system.

I conducted a simple visual analysis of the shell command data provided by Rich Hug's script which extracts data from the MDS CCC Harvest Tool. I have found that there are three types of elements; MDS Packages, Rollup Packages, and Work Packages. Rollup Package represents elements that are generated "top-down" through planning and requirement specifications [2]. This corresponds to the following Architectural Elemenets: Implementation Task Rollup, Implementation Task, Change Package Rollup, and Verification Package Rollup. MDS Package represents both "top-down" and "bottom-up," where "bottom-up" means that the elements are generated through defects and other problems. This corresponds to the following Architectural Elements: Requirement Pacakge, Change Package, Internal Anomaly Report, Internal Modification, and Verification Package. MDS Package is the unifying representation of both Rollup Packages and Work Packages.

Furthermore, there is a releationship between Rollup Packages and MDS Packages. Rollup Packages contain one or many Work Packages. Work Packages are the packages that contain the actual implementation and Rollup Packages are packages that contain Work Packages.

This finding has allowed me to better represent all Architectural Elements in the hackyJPLBuild system as either Rollup Packages or Work Packages, see MDS Packages, Work Packages, and Rollup Packages in Section 2.2 for a more detailed explanation of the internal representation of Architectural Elements in the hackyJPLBuild system.

4.1.2 Understanding the MDS CCC Harvest State Model

In the hackyJPLBuild sysem, I implemented an analyses which derives the MDS CCC Havest State Model to understand how the different MDS Packages move through the Harvest tool. This analysis calculates the number of occurrences of State Changes, in a specific time period, and

for a given MDS Package type, ie one of the Architectural Elements. The following figure provides an example of the MDS CCC Harvest State Model analysis.

Using the data provided by the previous analysis, I have created a graphical representation of the transitions. The following figure shows this graphical representation.

This figures shows several interesting results. First, 44% (76 of the 173) of Change Packages that leave the Dev Harvest State revert back to Dev at a later time. This shows that there is a considerable amout of Rework or unscheduled work. Second, 100% of Change Packages in the Integration Test Harvest State move to the Release Harvest State without any problems. Further implications or hypotheses of what these values indicate have not been determined at this time.

4.1.3 Validating the Raw Data

While conducting my analysis of the MDS CCC Harvest State Model, explained in the previous section, an interesting problem arose. The analysis shows a “mysterious State” Changes, the State Changes were not occuring in successive chronological order. The following figures shows this anomaly.

Based on this finding, I created a Hackystat analysis, whick finds all MDS Packages with the “mysterious” State Changes. The following figure shows an example of this analysis.

This analysis shows that 20% of all Change Packages have a” mysterious” State Change at some point in time. This indicates a major problem either two things; (1) the MDS CCC Harvest Tool or (2) the script which extracts data from the MDS CCC Harvest Tool. In any case this problem must be addressed.

4.2 Case Study 2 - Statistical Analysis of YTD Data

Case Study 2 includes analyzing the YTD data to identify thresholds for factors that are associated with certain measurements of the MDS system. Proposed factors include: number of developers, number of entanglements, number of files in a MDS Package, and the number of build failures. Propsed measurements include: throughput and age.

For example, I propose the creation of the “Throughput Threshold” analysis. This analysis will find the average throughput value for a certain type of a MDS Package. For throughput valudes that are above the average, I will calculate the threshold values by finding the averages of the factors. The following table shows an example of the threshold values.

Measurement: Throughput 7 Months Thresholds: Number of Developers = 2, Number of Entanglements = 6, Number of Files = 25, and Number of Build Failures = 4.

The previous example data shows the thresholds for several factors that influence throughput. For example, MDS Packages with 6 or more entanglements will have an above average throughput measurement, MDS Packages with 25 or more files will have an above average throughput measurement, and so on.

If analyses can identify threshold attributes of the MDS system then, predicting problems in the Harvest System should be relatively easy. The next section explains Case Study 3 - Predicting Problems.

4.3 Case Study 3 - Predicting Problems

If the previous case study can provide thresholds that indicate above average measurements, then Case Study 3 can investigate if these thresholds can predict future above average measurements. This case study will analyze real time data provided by Harvest.

The methodology for this case study is the following. For two months the threshold values for above average measurements will be stored in the hackyJPLBuild system. When a MDS Package contains a value beyond the threshold value, the system will record the package's unique identifier and other information about the package's current state. At the end of two months, I will conduct a study of the MDS Packages that were stored in the system to analyze if the threshold values predicted about average measurements.

If analyses can accurately predict above average measurements of the MDS system then, preventing problems is the next logical step. The next section explains how hackyJPLBuild could be used to prevent problems.

4.4 Preventing Problems

The next logical step, assuming that the three previous case studies are successful, is to apply the predictions in a prevention case study. Preventing porblems means that we must intervene with the build and developement process of MDS however, I feel that intervention is not an obtainable goal for the duration of this thesis project.

When the time becomes appropriate for hackyJPLBuild to intervene with MDS, hackyJPLBuild will have collected a significant amount of raw data and have created a significant amount

of thresholds and predictions. At this time, the hackyJPLBuild system can alert developers of potential problems that have been predicted. Developers can then, attempt to “bring the threshold value down,” or pay special attention to the potentially problematic MDS Package.

Bibliography

[1] Watts Humphrey. The IBM large-system software development process: Objectives and direction. *IBM Systems Journal*, 24(2), 1985.

[2] Philip M. Johnson. Improving the dependability and predictability of jpl/mds software through low-overhead validation of software process and product metrics. Technical Report CSDL-02-03, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, May 2002.

[3] K. H. Moller and D.J. Paulish. *Software Metrics: A practioner's guild to improved product development.* Chapman and Hall, Boundary Row, London, 1993.

[4] X2000 MDS: Technology Mission Data System. `<http://x2000.jpl.nasa.gov/nonflash/technology/mds.html>`.