Information and Software Technology 56 (2014) 183–198

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Data stream mining for predicting software build outcomes using source code metrics

Jacqui Finlay, Russel Pears, Andy M. Connor *

Auckland University of Technology, Auckland, New Zealand

ARTICLE INFO

Article history:
Received 5 December 2012
Received in revised form 4 September 2013
Accepted 5 September 2013
Available online 14 September 2013

Keywords:
Data stream mining
Concept drift detection
Hoeffding tree
Jazz
Software metrics
Software repositories

ABSTRACT

Context: Software development projects involve the use of a wide range of tools to produce a software artifact. Software repositories such as source control systems have become a focus for emergent research because they are a source of rich information regarding software development projects. The mining of such repositories is becoming increasingly common with a view to gaining a deeper understanding of the development process.
Objective: This paper explores the concepts of representing a software development project as a process that results in the creation of a data stream. It also describes the extraction of metrics from the Jazz repository and the application of data stream mining techniques to identify useful metrics for predicting build success or failure.
Method: This research is a systematic study using the Hoeffding Tree classification method used in conjunction with the Adaptive Sliding Window (ADWIN) method for detecting concept drift by applying the Massive Online Analysis (MOA) tool.
Results: The results indicate that only a relatively small number of the available measures considered have any significance for predicting the outcome of a build over time. These significant measures are identified and the implication of the results discussed, particularly the relative difficulty of being able to predict failed builds. The Hoeffding Tree approach is shown to produce a more stable and robust model than traditional data mining approaches.
Conclusion: Overall prediction accuracies of 75% have been achieved through the use of the Hoeffding Tree classification method. Despite this high overall accuracy, there is greater difficulty in predicting failure than success. The emergence of a stable classification tree is limited by the lack of data but overall the approach shows promise in terms of informing software development activities in order to minimize the chance of failure.

1. Introduction

Software development projects involve the use of a wide range of tools to produce a software artifact, and as a result the history of any given software development endeavor may be distributed across a number of such tools. Recent research in this area [1] has classified the types of artifacts that can be used to reconstruct the history of a project. These include the source code itself, source code management systems, issue tracking systems, messages between developers and users, meta-data about the projects and usage data. Software repositories such as source code management systems have become a focus for emergent research as being a potential source of rich information regarding software development projects. The mining of such repositories is becoming increasingly common with a view to gaining a deeper understanding of the development process.

Jazz (http://jazz.net) is a collaborative software engineering toolset developed by IBM that has been recognized as offering new opportunities in this area because it integrates the software archive and bug database into a single repository. This is achieved by linking bug reports and source code changes with each other [2]. In addition to the toolset itself, IBM has also released the data repository associated with the development of Jazz. This provides access to nearly all of the artifacts that can be used to construct the history of the project [1] and includes developer communication as well as source code and bug reports. Such a repository provides much potential in gaining valuable insights into the development process yet comes with particular challenges. One of the main challenges is that the Jazz environment is not designed to maintain a complete history of build events. The implication of this challenge is that mining the repository for a given project does not involve a static base of data which grows over time. Instead,

* Corresponding author. Tel.: +64 99219999.
E-mail address: andrew.connor@aut.ac.nz (A.M. Connor).

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.09.001

the size of the data is essentially fixed but the bounds of the data change over time meaning that the available data at any instant is just a snapshot of the total development history.

Traditional data mining methods and software measurement studies are tailored to static data environments. These methods are typically not suitable for streaming data which is a feature of many real-world applications. Software project data is produced continuously and is accumulated over long periods of time for large systems. The dynamic nature of software and the resulting changes in software development strategies over time causes changes in the patterns that govern software project outcomes. This phenomenon has been recognized in many other domains and is referred to as data evolution, dynamic streaming or concept drifting [3]. However there has been little research to date that investigates concept drifting in software development data despite being recognized as an area of interest [4]. Changes in a data stream can evolve slowly or quickly and the rates of change can be queried within stream-based tools. This paper describes an initial attempt to fully extract the richness available in the Jazz data set by constructing predictive models to classify a given build as being either successful or not, using software metrics as the predictors for a build outcome. In this context, a build attempt in the Jazz software is the process of performing software subsystem integration and the integration of multiple subsystems. Build success is therefore a successful integration which includes compilation, testing and packaging without producing any errors [5]. Build failure is therefore the presence of an error that prevents the creation of a deployable software package.

This research investigates the construction of predictive models to determine build outcomes in advance of the build attempt. Previous work [6,7] in this area has shown that there is potential for such prediction to occur, however the use of traditional data mining approaches results in unstable models with limited applicability. This work therefore investigates the application of data stream mining approaches that are capable of adapting to changes in the underlying data distribution to determine whether they produce more stable models. The goals of the research are formally expressed in Section 4.

The models that are built in this research involve the tracking of the trajectory of classification accuracy over time. The models are built incrementally as each new data instance (software build) that arrives is used to update the existing model. With the Jazz repository there exists a certain set of builds having known outcomes that can be chosen from to train and induce an initial model (in the form of a Decision Tree) and the rest of the builds can then be utilized to incrementally update the model. This implies that there is freedom to choose in which order the instances are supplied and hence this produces an opportunity to track the effects of instance ordering on classification accuracy of the predictive model that are constructed and incrementally maintained.

Section 2 provides a brief overview of related work. Section 3 discusses the nature of the Jazz data repository and software metrics that were utilized during the mining the repository. Section 4 discusses the approach to mining the software repository in Jazz, and initial results are presented in Section 5. Finally, in Sections 6 and 7, a discussion of the limitations of the current work and a plan for addressing these issues in future work are presented.

2. Background and related work

The mining of software repositories involves the extraction of both basic and value-added information from existing software repositories [8]. Such repositories are generally mined to extract facts by different stakeholders for different purposes. Data mining is becoming more prevalent in software engineering environments [4,9,10] and the application of mining software repositories include areas as diverse as the development of fault prediction models [11], impact analysis [12], effort prediction [13,14], similarity analysis [15] and the prediction of architectural change [16] to name but a few. The growth in popularity in mining software repositories have led some researchers to believe that we are on the brink of introducing the idea of Software Intelligence (SI) as the future of mining software engineering data [17]. Hassan and Xie [17] argue that "Software Intelligence offers software practitioners (not just developers) up-to-date and pertinent information to support their daily decision-making processes. SI provides practitioners with access to specialized fact-supported views of their software system so they can answer critical questions about it."

According to Herzig and Zeller [2], Jazz offers not only huge opportunities for software repository mining but also a number of challenges. One of the opportunities is the provision of a detailed dataset in which all software artifacts are linked to each other. To date, much of the work that utilizes Jazz as a repository has focused on the convenience provided by linking artifacts such as bug reports to specification items along with the team communication history. Researchers have focused on areas such as whether there is an association between team communication and build failure [5] or software quality in general [18]. Other work has focused on whether it is possible to identify relationships among requirements, people and software defects [19] or has focused purely on the collaborative nature of software development [20]. To date, most of the work involving the Jazz dataset has focused on aspects other than analysis of the source code contained in the repository and as such the full range of the available project history is not being fully utilised.

Research that focuses on the analysis of metrics derived from source code analysis to predict software defects has generally shown that there is no single code or churn metric capable of predicting failures [21–23]. However, evidence suggests that a combination of such metrics can be used effectively [24]. To date no such source code analysis in a data stream context has been conducted on the Jazz project data and it is the objective of this study to perform an in-depth analysis of the repository to gain insight into the usefulness of software metrics in predicting software build failure. In particular the focus is on using data stream mining techniques to enable software development teams to collect and mine SE data on the fly to provide rapid just-in-time information to guide software decisions as has been suggested in the literature [4]. However, this is just one element of a larger research agenda that looks to fully integrate the available project history into a usable decision support environment that supports the software development team. It is likely that such an approach will use not only software metrics, but also developer communication metrics [5] as well as repository level change measures. Such measures have been noted as being good indicators of failure in the literature [25].

Despite the claims that change measures are good indicators of failure, there is also evidence that fine grained source code change metrics are also good indicators of failure when compared to code churn metrics [26]. The conflicting evidence of what is likely to be a good indicator of failure implies that there is a need for a decision support dashboard that is based on multiple measures. The need for decision support in software engineering has been identified in the literature [27] and is clearly concordant with the concepts of Software Analytics [28] and Software Intelligence. The emergence of decision support models based around the use of software metrics is likely to see different aspects of software engineering research merge and overlap. This is already apparent with recent research that looks at using Search Based Software Engineering techniques as a feature selection technique applied to choosing software metrics for defect prediction [29]. Other approaches investigate the use of information density as a means of selecting