Toolkit for Automatic Collection and Predictive Modelling of Software Metrics

TOOLKIT FOR AUTOMATIC COLLECTION AND PREDICTIVE MODELLING OF SOFTWARE METRICS

BY

TIM JOHNSTON, B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2016) McMaster University

(Software Engineering) Hamilton, Ontario, Canada

TITLE: Toolkit for Automatic Collection and Predictive Modelling of Software Metrics

AUTHOR: Tim Johnston

B.Eng., (Software Engineering)

McMaster University, Hamilton, Ontario, Canada

SUPERVISOR: Dr. Douglas G. Down

NUMBER OF PAGES: viii, 157

Dedicated to my grandfather, who always kept himself busy

(but not too busy to laugh)

Abstract

For empirical software researchers, the management of experimental data and results is time-intensive and a significant source of error. As a result, research in this area is often difficult to reproduce, falsify and generalize. Software projects deemed feasible for study are typically of a limiting size and belong to particular domains for which measurement tools are available. We propose a toolkit for data gathering and exploratory modelling in empirical software research. Requirements are derived from issues commonly affecting studies of defect prediction and software evolution. We develop the toolkit by using a prototype-driven approach. The prototype toolkit is used to carry out a case study illustrating its effectiveness and potential for future extension.

Acknowledgements

I would like to acknowledge my supervisor, Dr. Douglas G. Down. His guidance has been crucial in refining my research interests into a feasible project. His teaching has given me a fresh perspective on software engineering.

I would also like to acknowledge my parents. They have always encouraged me to pursue education and to exercise lateral thinking.

Finally, I would like to thank my students and my peers in the RASSL research group. Their constructive criticism and requests for advice remind me that one never works alone.

This research was supported by the Natural Sciences and Engineering Research Council of Canada.

Contents

Abstract **iv**

Acknowledgements **v**

Contents **vii**

List of Tables and Figures **viii**

1 Introduction **1**

1.1 Motivating Literature 2

1.2 Related Projects 13

1.3 Organization of the Thesis 14

2 Problem Statement **15**

2.1 Scenario 15

2.2 Pre-Requirements 17

3 Requirements **19**

3.1 Data Requirements 19

3.2 Modelling Requirements 22

3.3 Non-Functional Requirements 26

4 Development Approach **30**

4.1 Prototype-Driven Development 30

5 Design **36**

5.1 System Architecture 36

5.2 Module Descriptions 37

6 Implementation **45**

6.1 Prototype Diagram 46

6.2 Prototype Components 47

7 Case Study **57**

7.1 Case Study Outline 57

7.2 Git Study Results 60

7.3 Results from other data sets 69

7.4 Discussion 75

7.5 Threats to Validity 76

8 Future Work **79**

9 Conclusion **84**

A Toolkit Source Code **85**

B Case Study Notebooks **107**

Bibliography **147**

List of Tables and Figures

5.1 Designed System Architecture 37

6.1 Prototype Toolkit Modules 46

7.1 Case study data sets 58

7.2 Model (1), assessment 61

7.3 Decision Tree visualization 62

7.4 Model (2), assessment 63

7.5 Model (3), file size 64

7.6 Model (3), file size (revised) 65

7.7 Model (3), cyclomatic complexity 65

7.8 Model (3), cyclomatic complexity classification 66

7.9 Model (4), binary classification 67

7.10 Model (4), multiple classification 67

7.11 Model (4), random forest 69

7.12 Vim Results 70

7.13 OpenSSL Results 71

7.14 httpd Results 73

7.15 nginx Results 74

Chapter 1

Introduction

Many researchers have attempted to apply the methods of empirical science toward understanding software quality. This typically involves experimentation with the processes and organizations which develop software, and the software artifacts which are created. Areas of interest include the certification of high-reliability systems, management of software evolution, and evaluation of the tools and techniques used by developers. For both academic researchers and industrial software engineers, there is a great deal of interest in studying data about software projects.

Although researchers set out to perform rigorous scientific studies of software, they often encounter issues in the gathering and representation of data, as well as the validity and generalization of results. The methods used to carry out studies are often inconsistent, and the large scale of many software projects leads to compromises in sample size and representativeness of the gathered data. Software-based automation has brought consistency to such research tasks in the past. Therefore, we view this as a software development problem.

This work chronicles the development of a toolkit which can be used by researchers and developers to carry out studies of software projects. We begin this chapter with a review of motivating literature and background material. Following this, we give an overview of similar projects. This chapter concludes with an overview of the chapters to follow.

1.1 Motivating Literature

Empirical software engineering (ESE) has been an active area of study for many decades. Many research goals have been represented by ESE, including defect prediction, development time estimation, team organization and the design of new programming language features. Our review of the literature in this area indicates a common set of fundamental issues occurring in most studies. We acknowledge the contributions of several authors who have attempted to clarify these issues, to provide guidelines for ESE researchers and to organize efforts toward further study. We begin with a review of the common topics and techniques in ESE.

Evidence-based Engineering

ESE research has considered many different aspects of software development, and has provided many powerful results. However, it remains disconnected from much of the work in the field of software engineering. Prominent researchers have attempted to strengthen this connection, and to guide studies toward stronger results and greater relevancy.

Many have noted the prevalence of untested assertions and dogmatic viewpoints in software development, calling for an increased emphasis on empirical study. In a foundational discussion of this issue, Tichy [1] reviewed the role of experimentation in science and addressed objections to experimentation in computer science. In a talk given at the Canadian University Software Engineering Conference, Wilson [2] connected this with the field of software engineering. Both authors describe the emphasis on empirical evidence which has been vital in development of the traditional sciences and branches of engineering, but which has been lacking in software development. The authors of [3] consider a future based on "the interaction between theoretical and experimental results". They propose a cycle of "modeling, experimenting, learning, and remodeling" which can be repeated in the advancement of our field.

In a recent survey [4], Begel asked software engineers which common research questions they would like to see studied. This included topics such as the effectiveness of development practices, measurements of productivity and reuse of components. Similar questions have been proposed by Wilson [5] and Meyer [6]. With many questions awaiting empirical answers, leaders in the field have provided advice for those conducting future research. Kitchenham and Pfleeger [7] reviewed methodological and statistical errors in ESE. By comparison with the methods of statistical research in medicine, they gave guidelines for experimental design, data gathering, analysis, presentation and interpretation of results. Basili [8] gave an overview of the components in experimental studies and the challenges unique to the study of software. He outlined a vision for the integration of academic and industrial results, stating that "the research and business perspectives of software engineering have a symbiotic relationship."

While the ESE field has continued to grow, some topics have always been strongly represented in its research. We continue with a review of particular topics which are relevant to our study.

Static Code Measures

A frequent topic in empirical software engineering (ESE) is the measurement of source code. This has its origins in the work of McCabe, who proposed the cyclomatic complexity measure of programs [9]. Given such a measure that is believed to indicate software quality (e.g. reliability, testability, maintainability), it is natural to study the strength of predictive relationships and the application of these results. Fenton notes in [10] the correlation between McCabe complexity and other code measures, such as program size. Larger programs tend to be more complex (in the sense measured by McCabe), and this may lead to confounding results regarding cyclomatic complexity. A variety of studies that measured the complexity and size of software modules had contradictory results. Hatton reviewed these in [11], and found that (contrary to conventional wisdom) smaller software components were less reliable than those categorized as 'medium-sized'. Of course, such a result alone is not conclusive. The reviewed studies had used as source data projects of a particular size, which had been implemented in particular programming languages. These and other factors often limit the results of ESE studies. Due to rapid growth in industry, the source code to be studied is continuously expanding to larger scales. The adoption of new development tools and techniques diversifies the projects available for study and complicates the work of researchers. Despite these challenges, empirical study of measures such as program size and complexity remains an interest of many software developers.

Language-agnostic Measures

Language-agnostic measurements have been proposed to mitigate the effects of language diversity in software development, and to reach for *fundamental* results about software measurement. Hindle [12] found that the indentation level of source code in a variety of languages is a valid proxy for McCabe complexity and Halstead maintainability [13]. This result is valuable as it allows a simpler and more general measurement to be made (without requiring the use of specialized tools). The result also indicates that there may be complexity intrinsic to indentation which is manifested in source code.

Change Measures

In addition to measures of static structure, researchers have studied the social factors impacting source code. Examples include the changes made by developers and the organizational structure of development teams. These studies are motivated by works such as Brooks' *Mythical Man-Month* [14]. Similar discussions by Conway [15] proposed the idea that systems take on the structural characteristics of the organizations which create them. Typically, this manifests itself in the creation of separate software modules by separate teams. A related study by Nagappan [16] found that the distance between developers in the organizational structure at Microsoft was a significant predictor of post-release defects. Although the ideas of Conway and Brooks have been studied extensively, they do not consider the fine-grained changes through which software systems evolve. In [17], it is noted that software change sizes often follow power law distributions. The authors proposed that source code changes may include mechanisms of preferential attachment and self-organizing criticality, which

are known to drive power law behaviour [18]. In quantifying such changes, a common measure is the number of lines added and deleted (known as *code churn*). A study of software reuse [19] found that code churn correlated more strongly with pre-release defect rates than static code metrics such as cyclomatic complexity. Another study of code churn by Nagappan et al. [20] found that measures derived from churn and size metrics can accurately predict defects. D'Ambros et al. [21] investigated the relationship between coupled changes (separate software modules which are often changed together by developers) and defects. They proposed in [22] a measurement of the extent to which authors share in the changes made to a module. Such measurements are valuable because they are language-agnostic and can capture social (rather than technical) drivers of software quality. Change metrics have proven to be effective in isolated studies, but they face the same issues of scalability and representativity as static code measures. These measurements require source code management (SCM) history data about the projects under study. Although this is available in many studied projects, it is not strictly built into the source code that constitutes developed software. This necessitates the creation and use of extra measurement tools, and further differentiates studies. Although software metrics are specific to this field, empirical software research also exhibits many of the issues affecting contemporary data-driven research in other sciences.

Research Challenges

Although many software attributes and measurement techniques have been proposed, researchers frequently struggle with applying them. In [3] and [23], both academic and industrial groups attempted to apply architectural models of software

reliability to large public data sets. A recurring issue was the gathering and preparation of data for study. The techniques used for this were specific to the studies themselves, and required significant manual labour. For the researchers in [24], a small sample of source code revisions was used to illustrate their hypotheses. It is not clear whether their conclusions can scale to larger sample sizes, or which practical techniques one could use to attempt larger studies. This reflects the typical ESE issues of ecological inference and ecological fallacy described by Posnett et al. in [25]. Software projects are often composed of modules organized in hierarchies, and researchers must be aware of the implications this has for their studies. Empirical measurement of software can include statements about aggregated components, which may not apply to disaggregated components, and vice versa. The large size of real-world projects, coupled with reliance on domain-specific measurement tools made the preliminary work in these studies difficult and time-consuming. This was not only an obstacle for the above researchers. It also serves to discourage others from carrying out further studies which could extend or falsify these results. Empirical software research has often been limited to isolated or unrelatable results, owing to issues in data collection and availability.

In addition to the issues facing ESE research, computationally-driven research in many fields has struggled with reproducibility and the publishing of experimental data. Contemporary scientists increasingly require training to develop the software necessary to explain or falsify their results. Organizations such as Software Carpentry [26] attempt to address this through extracurricular teaching. In a review of reproducible research discussions [27], Boettiger notes that for both cultural and technical reasons, the source code used to generate research results is typically not

published. This can be a barrier to peer review and replication. An ongoing effort by Krishnamurthi [28] attempts to reproduce computational environments and results from the software made available by resesearchers in computer science. As Boettiger describes in [27], software solutions to this problem (such as workflow systems and virtual machines) have been proposed but are not yet widely accepted. Along with issues of software development and publishing, ESE researchers struggle to identify and collect data for study.

Data Sources

Historically, difficulties in data collection have been addressed by providing curated data sets. Two such data sets are the SIR [29] and PROMISE repositories [30]. While this approach allows for repeated study of the same data, it does not account for the issues of scale and rapid advancement that characterize the software industry. Although both have been useful to researchers, they require maintenance by their providers. In the case of SIR, many of the provided datasets draw from 'toy' software applications which do not represent real systems. The quality and usage of some PROMISE datasets have been criticized for containing a large number of repeated samples [31]. In addition to curated repositories, researchers have pursued open source software projects as a source of data. In [32], Robles indicates why open source data is appropriate for study. Today, open source software dominates the infrastructure upon which the internet depends, including the Linux operating system, Apache web server, and OpenSSL security library. These projects have accessible source code management (SCM) repositories on the popular website Github [33]. Github and similar websites serve as an ever-evolving, highly-representative source of software

projects to be studied. They are publically available and can be accessed uniformly. Projects such as GHTorrent [34] and GithubArchive [35] have attempted to leverage this in creating public datasets which evolve automatically as their source material changes.

While new data sources have become widely available and applicable to ESE research, new techniques for modelling and analysis have been incorporated from other fields.

Machine Learning

The recent trend in computing toward data analytics has begun to impact ESE research, with many groups applying machine learning techniques to the field. These are domain-independent methods of exposing relationships in data, algorithmically modelling data and discarding measurements which are not explanatory. By employing such techniques, software researchers stand to benefit from strong results in other fields such as biology, medicine and finance. Machine learning models differ fundamentally from those which attempt (a priori) to capture the relationships between software attributes and desired qualities. Where many studies invent a model and fit data to it, machine learning algorithms generate candidate models which one may fit data to. A common analogy is that model application uses a program and its input to generate output, while machine learning begins with inputs and outputs, and ends by obtaining a suitable program. It is important to note that machine learning does not attempt to generalize and explain the true mechanisms behind phenomena. However, it can be used to direct researchers toward data relationships which may yield results useful to other forms of study. Since these techniques are widely available to

contemporary software developers, they represent a common point of reference for industrial and academic ESE research, as proposed by Basili [8]. We give a brief review of machine learning terminology which is used throughout the remainder of the thesis.[1]

Machine learning encompasses a variety of tasks which process and model data. For ESE researchers, modelling is typically used for supervised learning tasks such as classification (the prediction of discrete categorical variables) and regression (the prediction of continuous variables). Data sets are organized as arrays of samples. Each sample includes features (explanatory variables) and targets (response variables). Many algorithms have been designed to support these tasks, each with its own requirements for data and differing levels of complexity. Examples include linear regression, K-nearest neighbours and support vector machines. Our work focuses on decision trees[2] and related approaches such as random forests. Trees model predictions by a series of 'yes-no' questions (decisions). As one traverses the tree, decisions are made and the sample data is filtered. Eventually, at the leaf nodes of the tree, sample data is mapped to classification or regression values. Among the canonical machine learning models, trees are known to be simple and intuitive. They match the approach taken in typical decision-making tasks such as the classification of species and medical diagnoses. Unlike some other models, they do not require the input data to be pre-scaled to achieve best performance. When the values used to build decision trees are unscaled, they represent the original ranges of measured quantities. This makes them simple to interpret - each decision groups samples by values in the measured range of a feature. Although machine learning models can give valuable

[1] A detailed visualization of the reviewed terms is available at [36].
[2] Tree-based prediction models are summarized in [37].

predictions and insight into data, they must be carefully validated for use.

Machine learning models (and statistical models in general) must be used with care. They attempt to model the distribution of the given data, which may not reflect the more general reality from which samples are taken. For prediction models, this manifests itself in erroneous outputs such as false classifications and inaccurate regression values. As with other statistical techniques, these problems can often be mitigated by the gathering and use of more sample data. In cases where larger sample sizes do not lead to improved models, the methods used may be at fault. To validate a model with sample data, one typically fits the model to a *training* subset of the data. The trained model is used to make predictions about a *testing* subset, and these are compared with the ground truth values. This approach allows one to establish the accuracy and usefulness of the trained model, and can be extended to accommodate a wide variety of assessments. In *cross-validation*, one repeatedly trains and tests models on different subsets of the data. This can reveal weaknesses in modelling that a single train-test split cannot. For example, if the modelling approach used is sensitive to the subset of data that is used for training, the trained models may not generalize to future data. Although there are many approaches to assessing models, one must also be able to improve the accuracy and generality of models. This echoes a famous quotation by the statistician George Box: "Since all models are wrong the scientist cannot obtain a 'correct' one by excessive elaboration."[38] Box went on to say that "the ability to devise simple but evocative models is the signature of the great scientist," while "overelaboration and overparameterization is often the mark of mediocrity."[38]

This has a direct analogy in machine learning algorithms. In this area, the role

of the 'scientist' is partially taken by a model construction algorithm. More complex models are prone to overfitting. They describe the training data very accurately, but do not generalize well on the testing data or future data. For tree-based models, this usually denotes trees that are very deep or broad, or have many leaves. Similarly, models which are too simple to describe the training data are said to exhibit underfitting. It is the role of human beings using machine learning algorithms to guide them and refine the results, to control phenomena such as overfitting and underfitting. In addition to its effect on prediction accuracy, model complexity also impacts interpretability. One may find it difficult to explain a complex model to outsiders, or to extrapolate actionable rules. Of course, the value of interpretation depends on the context in which models are used. Next, we review machine learning applications in a variety of ESE studies which have inspired our work.

Pinzger et al. [39] used machine learning models to predict software changes of various types. They carried out a two-stage analysis process, identifying which of the measured variables exhibited strong correlations and then constructing classification models using those variables as input. They noted that their methods could be applied as part of the continuous integration processes used by many contemporary developers. In other words, just as one creates 'nightly builds' of developed software, one could create 'nightly models' of software quality which evolve in parallel with the products they represent. In [40], machine learning models were used to predict dynamic software attributes (such as heap memory usage and thread pool consumption). The studies in [39] and [40] tested a variety of model construction algorithms against their gathered data, with the goal of identifying model types which are consistently useful in this domain. It is common in data science to explore various standard

modelling options in the context of the studied data. The work in [41] attempted to identify which of the canonical machine learning models are most applicable to several types of software data. They employed a cross-validation technique and ranked the employed modelling algorithms by the accuracy of their results. Together, these studies illustrate important advantages of the machine learning approach. It can be integrated into other workflows, and is amenable to a variety of data sources and modelling techniques. It allows data and models from ESE to be compared with those created in other fields. We wish to note that these studies do not explicitly consider the interpretability of their models. They judge their models on predictive accuracy alone, which may be appropriate to their task of creating automated decision support systems. As we search for comparable results in academic and industrial ESE studies, interpretable models may be more valuable than high-performance models. A model which makes poor predictions (arrived at by sound experimental methods) can still be useful to challenge or invalidate previously-held beliefs about software development.

1.2 Related Projects

Other researchers have worked on similar problems. In particular, Fursin [42] identifies many of the same issues facing experimental research in computer engineering. Our approach is similar to that of Fursin, in that we work toward an extensible framework for such research. Workflow support systems such as Kepler [43] have been developed to address similar issues in contemporary research. Our approach is specific to the domain of software measurement. It is focused on data about software project repositories and software metrics. Outside of academic research, there is a strong interest in software management tools. Companies such as Empear [44]

and Bitergia [45] create user-specific models of software project data, enabling their customers to improve development operations and product quality.

The above authors have explored the possibility of automated data collection and research systems. Our work represents another view of this area, specific to empirical software engineering.

1.3 Organization of the Thesis

In Chapter 2, we describe a hypothetical situation which captures the motivating research issues. Following this, Chapter 3 gives the requirements of a software toolkit applicable to such a situation. Given the requirements, we describe in Chapter 4 the approach used to develop the toolkit. This approach guides the design, implementation and application of the system with an emphasis on prototyping and iterative refinement. Chapter 5 describes the finalized architectural design of the system. In Chapter 6, we detail the implementation of a prototype satisfying the design. To illustrate the qualities of the toolkit and of our development approach, we carry out a case study. Chapter 7 describes an application of the tool to study several types of software quality data across several source projects. Through reflection on the case study, Chapter 8 identifies areas for future extension of the toolkit. We conclude with a summary of the problem and its ongoing solution in Chapter 9.

Chapter 2

Problem Statement

In this chapter, we describe a hypothetical situation capturing the problem we wish to address, from the perspective of its academic and industrial stakeholders. The software development efforts following this chapter are intended to address this problem. A case study resembling the scenario is later used to illustrate the effectiveness of the developed software.

2.1 Scenario

This scenario describes the proposal of an ESE research study, with needs that can be addressed by a software support system.

Company X has been developing a software product using the C programming language. Their project consists of many source code files stored in a Git repository. Many authors have committed many changes to this project, gradually adjusting the source code over several years. Company X is aware of studies which have predicted large maintenance efforts and defects based on the size of software modules and the

size of changes to modules. Company X is concerned that some source files may be growing too quickly or growing to a critical size. It is believed that some of the modules have very complicated code which is difficult to test and may contain dangerous bugs.

Company X would like to gather data about the form of their source code files and their changes over time. From this data, they wish to predict the future size of modules and source code changes. Their developers believe in creating small modules with few responsibilities, containing simple source code. After identifying modules which are too large or complex, the company will allocate developers to refactor them.

The developers at Company X know that software maintenance is an ongoing task. They want to have an automated system to repeat this study again in the future.

The research division of Company X is also interested in the outcome of this work. They would like to compare the study of their product against other C programs stored in Git repositories. They know the scientific community could benefit from access to many similar studies.

Some engineers at Company X have been reading about newly-proposed software metrics and they intend to measure these on their source code in the future. Could these measurements provide new insight into the models of size, complexity and change?

Everyone at Company X agrees that it would be unwise to make decisions using untrustworthy models. They want a system that can assess the models it creates, to accept those which are useful and reject or refine those which are not.

2.2 Pre-Requirements

Based on our literature review and the scenario, our problem has several important aspects. They are summarized as follows:

- Researchers wish to study static code attributes (both language-specific and language-agnostic).
- Researchers wish to study changes in the revision history of software.
- Measures of software quality can be viewed as both explanatory and response variables.
- Developers wish to identify modules having certain qualities in order to make targeted changes.
- There may be correlation between the measured attributes, including confounding effects from related measurements.
- The scale and lack of uniformity between software projects makes data collection tedious and error-prone.
- Researchers wish to compare software projects in many contexts (scale, programming language, domain of application, execution platform).
- New measurements of software quality are frequently proposed and would benefit from empirical assessment.
- There is a wealth of open source software available for study.

- Studies may or may not begin with a statement of the hypotheses to be tested. Researchers may wish to explore relationships in data as possibilities for controlled study.

- Machine learning techniques provide a uniform, flexible approach to such exploratory modelling.

In the next chapter, we describe the requirements for a software automation system applicable to each of these aspects of the problem.

Chapter 3

Requirements

In this chapter, we give the requirements of the system to be developed. Some requirements are derived from difficulties encountered in previous studies, challenges to existing results and barriers to repeatable research. In addition to these requirements, we define some that afford extension and generalization of the tool in future research.

We begin by detailing the functional requirements. These are separated into Data Requirements and Modelling Requirements. Following this, we describe the Non-Functional Requirements.

3.1 Data Requirements

Data Requirements pertain to the acquisition, preparation and storage of measurements about software artifacts under study. A full workflow would carry out tasks in each of the following stages.

Acquisition

D1: The user must be able to choose a project for study.

D2: The system must be able to measure source code entities from the chosen project at all points in their recorded history.

D3: The system shall store all acquired data in a repository dedicated to the project under study.

D4: The system must be able to acquire SCM revision history data from the chosen project.

Rationale: Empirical software studies often compare measurements of separate software projects. A uniform representation of gathered data is important for such studies.

SCM repository data serves as a feasibility requirement for the study of software projects. This represents an interface to the content of software artifacts and their changes over time.

D5: The system must be able to acquire change metrics data from the chosen project.

D6: The system must be able to acquire static code metrics data from the chosen project.

D7: The system must be able to interface with third-party tools to gather data.

Rationale: Change and code metrics are common candidates for empirical software research. There are many existing metrics tools which can be used to provide these and other measurements. This allows us to limit the scope of the system under development.

Preparation

D8: The system must be able to identify invalid or missing values in the data gathered from measurement tools.

D9: The system must be able to replace the identified invalid or missing data with values suitable for modelling.

D10: The system must label each data sample with the represented *time* as a feature.

D11: The system must label each data sample with the represented *entity name* as a feature.

Rationale: This stage of research work is often a large time investment. Errors and non-uniformity at this stage (arising from ad hoc methods) can lead to weaker or less useful research results.

Since the system is concerned with modelling software artifacts over time, the *time* and *entity name* features are used to group prepared data samples for study. This allows for the variation of granularity in data sets. Times could be represented with fine-grained SCM commit dates, or by aggregation (months, quarters and years). Entity names can refer to particular source code files, or aggregates such as packages, libraries, and binary applications.

Storage

D12: The system must be able to store prepared data sets for future re-use.

D13: Stored data sets should be in the form that they are gathered from data sources in the Acquisition stage.

Rationale: Some data-gathering tools may have significant running times, but they ultimately deliver information in the same form as faster tools (samples of feature data). Storing data can save time in future research tasks and improve reproducibility. Additionally, stored data can serve as a point of interoperation between this and other systems.

Data is stored after it is acquired, but before it is prepared for modelling. For this to be effective, the preparation and modelling steps should be faster than the data acquisition step. This decouples the data-gathering workflow from the modelling workflow.

3.2 Modelling Requirements

Modelling Requirements pertain to the construction, assessment, interpretation and refinement of models based on data.

Feature Engineering

M1: The system must be able to generate derived features from the gathered data. For example, one can add a feature respresenting the net churn (the difference between 'lines added' and 'lines deleted').

M2: The user must be able to choose which derived metrics are added to the feature data.

Rationale: Many studies work with features that are functions of the basic features given by data gathering tools.

M3: The system must be able to merge metrics data tables via common keys given in the Preparation stage (entity names and times).

M4: The user must be able to choose which data sources to merge from. This must provide the generated feature data for use in other tasks.

Rationale: Many studies test for the relationship (or lack thereof) between different types of measurements.

Response/Target Selection

M5: The system must be able to transform features into classification targets using a binary category generating function. For example, this may categorize data samples by the predicate 'has net churn less than 500 lines'.

M6: The system must be able to transform features into classification targets using a multi-category generating function. For example, this may categorize data samples into bins 'low churn', 'medium churn', 'high churn'.

M7: The system must be able to designate features as regression targets. For example, this may include 'file size in lines'.

M8: The system must support modelling multiple response variables simultaneously. For example, this allows one to create a model regressing on both 'file size in lines' and 'net churn'.

M9: The user must be able to choose which of the above response selection options are to be used. This must provide the generated response variable data for use in other tasks.

Rationale: Typical machine learning applications assume detailed knowledge of the response (or target) variables. For the research toolkit, this is not the case. We wish to support research workflows which are ignorant (a priori) of which response variables to model and how to define classification or regression targets. Users can explore various options for supervised learning, and discover which data relationships hold (or do not hold) for their datasets.

By decoupling the model type and accuracy measurements from the response variable, we enable users to search for nonlinear effects which may not be revealed through simple measures of correlation.

Construction

M10: The user must be able to choose the machine learning model to be trained from a variety of standard types.

M11: The user must be able to adjust the training parameters of the chosen model.

M12: The system must be able to fit the supplied data to the chosen model. This must provide the fitted model for use in other tasks.

Rationale: Researchers should be able to test the sensitivity of their hypotheses to the type of model constructed and its parameters. In typical machine learning applications, these choices are very amenable to user-driven adjustment and inquiry. However, such adjustment can also be automated to optimize for a given target such as cross-validation accuracy.

Assessment

M13: The system must be able to provide standard scoring measurements for fitted models.

M14: The system must be able to carry out cross-validation of the chosen model.

M15: The user must be able to choose the parameters of cross-validation. For example, this may adjust the contents of the training and testing data sets.

M16: The system must be able to carry out appropriate cross-validation techniques for time-ordered data.

Rationale: The accuracy scores of fitted models may indicate the strength of data relationships, peculiarities of the data gathered, or flaws in the research workflow. Cross-validation explores the sensitivity of the chosen model to the choice of training and testing datasets. By providing these measures, we allow users to compare results from the system on a uniform basis.

Interpretation

M17: The system must be able to report fitted model attributes. For tree-based models, this may include the features which have the greatest impact on sorting samples. For linear models, this may include the features which have the largest coefficients.

M18: The system must support visualization of models where applicable.

Rationale: Fitted models may use only a small subset of the features contained

in the training data. After model construction establishes what can be modelled, a natural direction for inquiry is how it is modelled.

Some model types are available for visualization. This can be valuable when presenting results, and can also serve as an approximate measure of model complexity (in the case of decision trees).

Refinement

M19: The system must be able to select the important features used by a fitted model, and reduce a given data set to only include the important features.

M20: The system must be able to re-train a chosen model on a reduced feature set.

M21: The user must be able to control the parameters of feature selection and re-training. This allows the user to make choices about interpretability and model complexity.

Rationale: After the interpretation stage indicates which features are important to the model, one may wish to discard unimportant data or train a simpler model. In particular, the adjustment of model parameters can be used to control overfitting or underfitting in the refined model. Refinement allows users to simplify future data collection efforts and to construct more general models.

3.3 Non-Functional Requirements

The non-functional requirements characterize the form of the toolkit. These are satisfied by the architectural choices made in the design.

Workflows

NF1: Data and Modelling services provided by the system should be offered to the user through independent interfaces (similar to an API or library).

NF2: Services must be open to combination in user-constructed workflows.

Rationale: Each user may benefit from different aspects of the system and may not require use of the complete workflow. By decoupling individual workflow stages, we allow researchers to begin using the system at any stage and continue iterative refinement of their results until they choose to stop.

Maintainability

NF3: The developed services should favour simple and general operations over complicated and domain-specific operations.

NF4: The developed system should include as little newly-developed software as necessary. The system should strive to re-use existing modules and simplify data processing tasks via high-level programming language features.

NF5: The source code implementing the system should be written in a self-documenting style. The programmer should strive to use descriptive names and human language constructs, rather than additional comments to explain unintuitive code.

Rationale: The realized system should be open to inspection, critique and adjustment by its users. Development of the system must be feasible within the timeframe given to this project.

Portability

NF6: The system shall support the major desktop operating system platforms (Linux, Mac OS, Windows).

NF7: The system should be made available for reuse without requiring users to rebuild source code or manage dependencies on third-party systems.

Rationale: The realized system should strive for reproducible computation, giving identical results when applied in a variety of host environments, without requiring the user to carry out inessential setup tasks.

Usability

NF8: User controls should be given sensible, widely-applicable default values.

NF9: Optional features and customizations should be hidden from users in the simplest use of the system.

Rationale: Customization should be possible but unnecessary, so that new users can gain valuable experience with the system quickly.

Performance

NF10: Data processing tasks should strive to make use of performance-optimized libraries.

NF11: The system should isolate access to third-party data sources from other workflow stages. Developers should strive to incorporate data sources into the toolkit for

improved performance and interoperation.

NF12: Default user control values should create models of low complexity.

Rationale: The processing of software project data can grow to prohibitively large scales. Users should understand the consequences of carrying out large studies before embarking upon them. The implementation of the system should not be a significant factor in the running time of data workflow tasks.

Simple models are often easy to train, test and interpret. Tradeoffs between model complexity, generality and interpretability should be made gradually by users after gaining familiarity with the system.

Although the toolkit requirements are now stable, they were revised throughout the project along with design and implementation artifacts. This resulted from the application of an exploratory development process. We proceed by describing this process as it relates to research-oriented software development.

Chapter 4

Development Approach

This chapter describes the software development process used throughout the project. We detail its distinguishing features, goals and areas of application. With this project as our example, we show that a novel approach to software development can be effective in research.

4.1 Prototype-Driven Development

Software projects are typically organized to fit a particular development approach. Examples include Test-Driven Development [46], Waterfall [47] or Spiral development [48], and various Agile methods [49]. Although there is disagreement about the relative merits and empirical validity of these approaches, they represent an important point of distinction between organizations. Throughout this project, we followed a distinct approach which is characterized by frequent prototyping and the refinement of prototypes into other development artifacts.

Prototype-driven development (as described here) is structured around the creation and execution of prototype software. This is in contrast to other methods which de-emphasize the realized software, in order to focus on articulating requirements and design goals. While agile methods may prioritize the creation of executable source code over documentation, they do not have explicit goals for the code which is created. It is assumed that code written throughout development will form the end product, and that developers begin the project with the technical skills required to complete it. A common goal of these and other methods is to reduce the cost and risk associated with software development.

Our approach is motivated by the needs of research-oriented (as opposed to business-oriented) software development. It may be appropriate when addressing the following challenges:

- The project requires developers to learn many unfamiliar domain concepts. For us, these come from the fields of software metrics, data science and scientific computing.
- The domain concepts are already well-represented by reusable software modules.
- Developers must explore the domain (including existing software) and identify areas for novel contribution.

In developing research software through a business-oriented approach such as the Waterfall model, one could identify unnecessary requirements and spend time 'reinventing the wheel'. For instance, many projects in this area involve the creation of metrics tools for which publically available alternatives exist. This is also true for many areas of scientific computing. High-quality implementations of machine

learning algorithms and data representation schemes have already been developed. By re-using these, we focus our work on aspects of the toolkit which are not provided by other packages and for which a standard design style may not be identified.

The Approach

Although we do not give a full account of prototype-driven development, we describe its key stages as follows:

- **Assemble a prototyping environment appropriate to the domain of application.**

  This acts as a staging area for the developer to carry out experiments which explore the problem space.

  Our environment was comprised of IPython notebooks [50], which are described further in Chapter 7.

- **Locate sources of domain-specific reference material.**

  This may include tutorials, blog posts, journal papers, conference talks, and API documentation. These aid in the creation of prototypes.

  In this project, valuable resources included conference talks from PyData [51] and SciPy [52], as well as notebook-style lessons from DataSchool [53]. API documentation from scikit-learn [54] and pandas [55] was frequently consulted during development of the toolkit.

- **Attempt to recreate common tasks from the domain within the prototyping environment.**

  This may include the mapping of domain concepts to the problem at hand. In our case, we created prototypes responsible for data gathering, model construction and model assessment.

  It may be the case that some aspects of the problem are not addressed by the domain concepts or available reference material. Developers identify lessons from this exercise which can be understood as requirements of the system to be developed.

- **After articulating requirements and design goals based on exploration, refine the prototype software into a satisfactory module.**

  This includes tasks typically called refactoring, such as generalization, replacement of particular values with exported parameters and abstraction to a common interface. A discussion of these and other patterns is available in [56].

- **Remove from the prototype environment any code which has been refactored into a module.**

  In the prototype, the code which has been refactored should be replaced with calls to the refactored module.

Development proceeds by iteratively carrying out the above stages (not strictly in order). Requirements and knowledge of the domain are extracted from prototypes, and prototype code is refined into stable code satisfying the emerging design. The prototype environment grows to resemble a top-level software module which uses the developed system to carry out the tasks of users.

Development Goals Addressed

Prototype-driven development allows developers to quickly identify what is already available for re-use in the domain of their project and what must be developed. It is amenable to other approaches which create traceable design artifacts. For example, prototyping exercises and their lessons can complement stakeholder discussions in tasks such as requirements elicitation. Prototyping in this manner can be employed as one of many valuable development tools. It should not be carried out dogmatically in place of other important approaches. We highlight several software development goals identified by North [57] which are addressed by this approach.

- **Spike and Stabilize**: We set out to create prototypes knowing that they will either be useful (to be refactored into production-quality code and design information), or be deemed unworthy and discarded. This decision results from the lessons learned during prototyping. It is an investment in learning and discovery (rather than product quality) in the short term. This is appropriate when development and execution of software can be repeated quickly (for instance, when using high-level programming languages and sources of feedback such as read-evaluate-print terminals or IPython notebooks).

- **Short Software Half-Life**: By evaluating prototypes and learning from them, we retain code and design concepts which are useful and discard those which are not. As we refactor prototype modules and re-use them to support future prototypes, we build confidence in the refactored modules and the design they represent. They are stabilized and their value is demonstrated.

- **Fits In My Head**: This is similar to the '7±2 rule' of design, which has its origins in the study of human short-term memory [58]. Developers can focus on a small number of details which characterize their prototypes. By refactoring successful prototypes into simpler modules, we remove sources of accidental complexity in the implementation. The stable software becomes more similar to the designed system, as the prototype has its detailed components replaced with connections to the refactored modules. The designed software is decomposed into a hierarchy of modules which is acceptable to users and understandable to developers.

This approach allowed us to refine the system architecture and a working prototype while developing expertise in the software measurement domain. We proceed by describing the stable design and implementation of the toolkit.

Chapter 5

Design

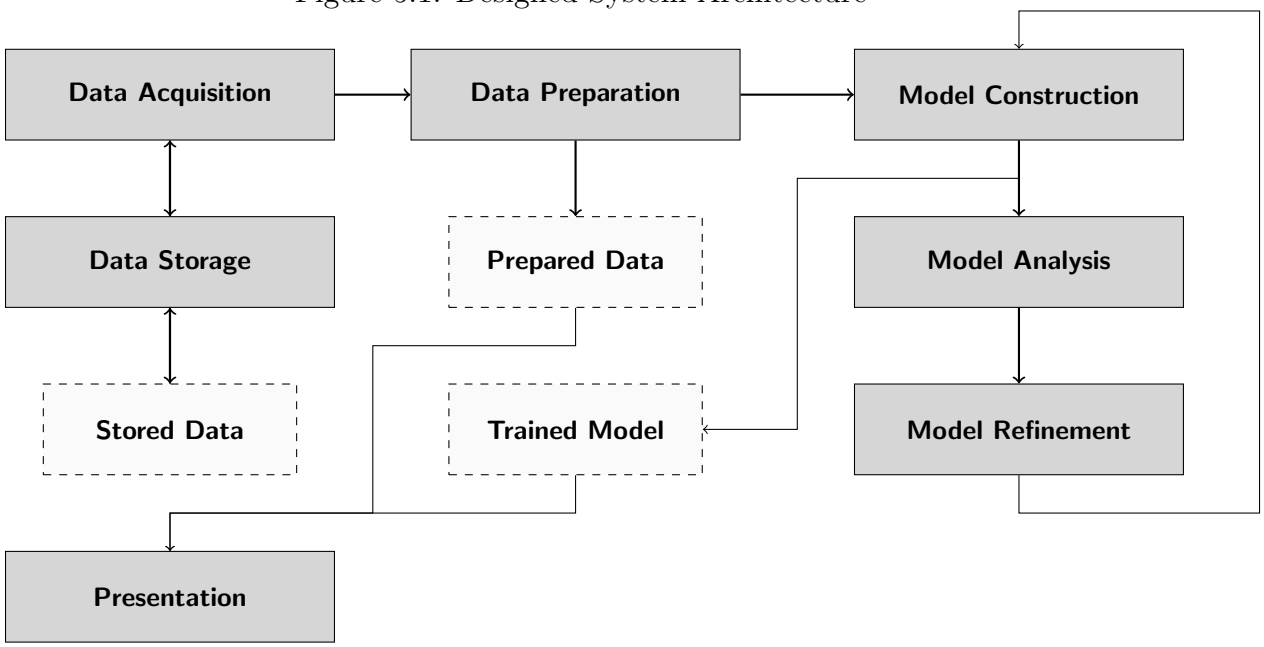
We describe the architectural design of software modules and their connections which satisfy the system requirements. The system seeks to automate tedious work and encourage user-driven inquiry. Therefore, each module satisfies some requirements by automation and some by offering user controls. Automated activities lead to uniformity between uses of the system, while user controls allow for individual exploration and extension of results.

We begin with a diagram illustrating the architectural connections between modules. Following this, we present a description of the designed modules.

5.1 System Architecture

The designed modules group responsibilities into workflow stages as shown in Figure 5.1. Shaded boxes represent modules, and dashed boxes represent data generated by usage of the modules. Solid lines indicate data flows between modules.

Figure 5.1: Designed System Architecture



5.2 Module Descriptions

We describe the designed modules by their features, implementation choices, inputs and outputs.

Data Acquisition

This group of modules is responsible for gathering data from third-party tools and local sources. This includes interfaces to:

- **SCM revision history**

  This provides data about the time periods available for study and the evolution of tracked software artifacts over time.

- **Change metrics**

  These measure the evolution of tracked software artifacts over time. Metrics include the number of changes made, the size of changes and the corresponding authors.

- **Static code metrics**

  These measure the form of software artifacts at particular moments in time. Metrics include file size, level of statement nesting and indentation level.

Data Preparation

This group of modules is responsible for preparing the gathered data for modelling.

Data Merging

This module is responsible for merging related data sets from various sources. This is divided into two steps:

- **Aggregate all data sources for one time point**

  For this time point, all of the measured features form one sample. The resulting dataset represents all of the features measured on entities at this time point.

- **Merge all aggregated datasets from time points selected for study**

  The resulting dataset represents all of the features measured on entities at all of the measured time points.

Feature and Target Engineering

This module is responsible for tasks which manipulate the explanatory and response variables under study (respectively, features and targets). The functions offered include:

- **Label-encoding features**

  This allows features such as timestamps to be represented as ordered integers, to encode ordered time in models.

- **One-hot encoding features**

  This allows features such as entity names to be represented as groups of one-hot features (binary categorical features which are 1 for the represented samples, and 0 for others). This enables the representation of named source code artifacts which are repeatedly measured over time, without the implication of meaningful order between names.

- **Adding derived metrics as features**

  This allows measurements to be combined into features of interest. This includes domain-specific transformations of software metrics (e.g. using lines added and lines deleted to calculate net churn) and domain-independent transformations from unsupervised machine learning tasks (such as clustering or dimensionality reduction).

- **Generating derived target variables**

  This allows users to explore possibilities regarding which target variables can be modelled, and the sensitivity of models to the target variables. Users can

begin with some idea about what they would like to study, and make statements about the data relationships which are exposed.

- **Scaling feature data**

  This allows feature data to be manipulated for application by machine learning algorithms. Some models require the training and testing data to be scaled so that it has zero mean and unit variance. However, some other models (such as decision trees) do not require this. Scaling can make results more difficult to interpret, since the absolute values of quantitites (such as net churn) are changed. When scaling has been applied, an extra layer of software automation can transform scaled values into their equivalent unscaled values. Scaling transformations are available for use in all workflows, but discretion is required on the part of the user.

- **Replacing missing or invalid data**

  This allows the preparation and modelling stages to address problems with the acquired data. Some feature data may be invalid, due to issues with third-party metrics tools or with the source files under study. A variety of different approaches can be taken, depending on the goals of the user. For example, one may wish to remove from the data set all samples containing invalid feature data. Alternatively, one may wish to replace invalid feature data with the mean or median of the valid samples.

Data Storage

This module is responsible for storing the gathered data. This includes several functions used throughout the system:

- **Database creation**

  This is performed automatically when gathering data about a new project under study.

- **Read a data table from the database**

  This is performed automatically when gathering data which has been previously stored in the database.

- **Write a data table to the database**

  This is performed automatically when gathering new data about the project under study.

Model Construction

This module is responsible for training models on prepared data.

- **Fit a model**

  Given an untrained model instance and a training data set, the model is trained and returned for use by other workflow stages.

Model Analysis

This module is responsible for evaluating trained models. This includes reporting and cross-validating model attributes given testing data.

- **Model attributes**

  Given a trained model, the toolkit is able to automatically measure and report several standard prediction model attributes. Classification models are evaluated on their Precision, Recall, F1-score and ROC AUC. Regression models are evaluated on their goodness of fit (e.g. R^2 error). We describe the classification metrics in the context of information retrieval (a typical binary classification task). Precision is the proportion of the *retrieved* results which are *relevant*, while Recall is the proportion of the *relevant* results which are *retrieved*. F1-score is the harmonic mean of Precision and Recall. ROC AUC describes the tradeoff between true positive rate and false positive rate as the classification threshold is changed. We refer to [59] for a detailed discussion of classification metrics.

- **Time Series Cross-validation**

  Given a model and a data set representing a time period, the toolkit can perform a cross-validation of the model with ordered times. This is distinct from typical cross-validation techniques which assume that all data samples are completely independent (and therefore not ordered by a feature such as *time*). To perform cross-validation with ordered times, we implement a technique described by Pochetti in [60]. The model is iteratively fitted to training sets which account for more of the available time points. Each trained model is evaluated on the applicable attributes. Cross-validation results allow users to determine the sensitivity of their models to the quality of their data over time. For example, if a regression model has abnormally high error in one time period, the data from that period may represent an outlier (exposing a caveat of the trained model,

or weakness in the data relationships it assumes). Developers may seek models which perform well using smaller training sets. These require less real-world time to make data available before they provide useful predictions. Models which represent a long time scale may be of interest to researchers. For example, it could be hypothesized that some phenomena of software evolution do not occur in short time scales.

Model Refinement

This module identifies the most important features used by a trained model. It re-trains the model using only the important features and compares the updated model with the original.

- **Feature selection**

  Given a trained model, this identifies the features which are most important. For tree-based models, this may include the features which have the greatest impact on sorting samples. For linear models, this may include the features which have the largest coefficients.

- **Model update**

  Given a set of important features, a trained model and the training data set, this updates the model. This includes a transformation of the training data set to include only the important features and re-training the model using the transformed data.

- **Model comparison**

  This allows the user to compare the updated model with the original model, highlighting differences in complexity, feature set size and interpretability.

Presentation

This module formats data and modelling artifacts for presentation.

- **Data visualization**

  This allows users to show graphically the relationships between gathered data. This may include typical techniques such as paired scatterplots and histograms.

- **Model visualization**

  This allows users to graphically represent trained models (when applicable). This may include diagrams of tree-based models, or plots of 'nearest neighbour' models showing decision boundaries.

We proceed by describing an implementation of the designed modules and their architecture.

Chapter 6

Implementation

We detail the implementation of a prototype system satisfying the design. As data sources, we use a mixture of third-party tools and original software modules. The prototype takes advantage of well-established libraries for tasks such as data representation and model construction.

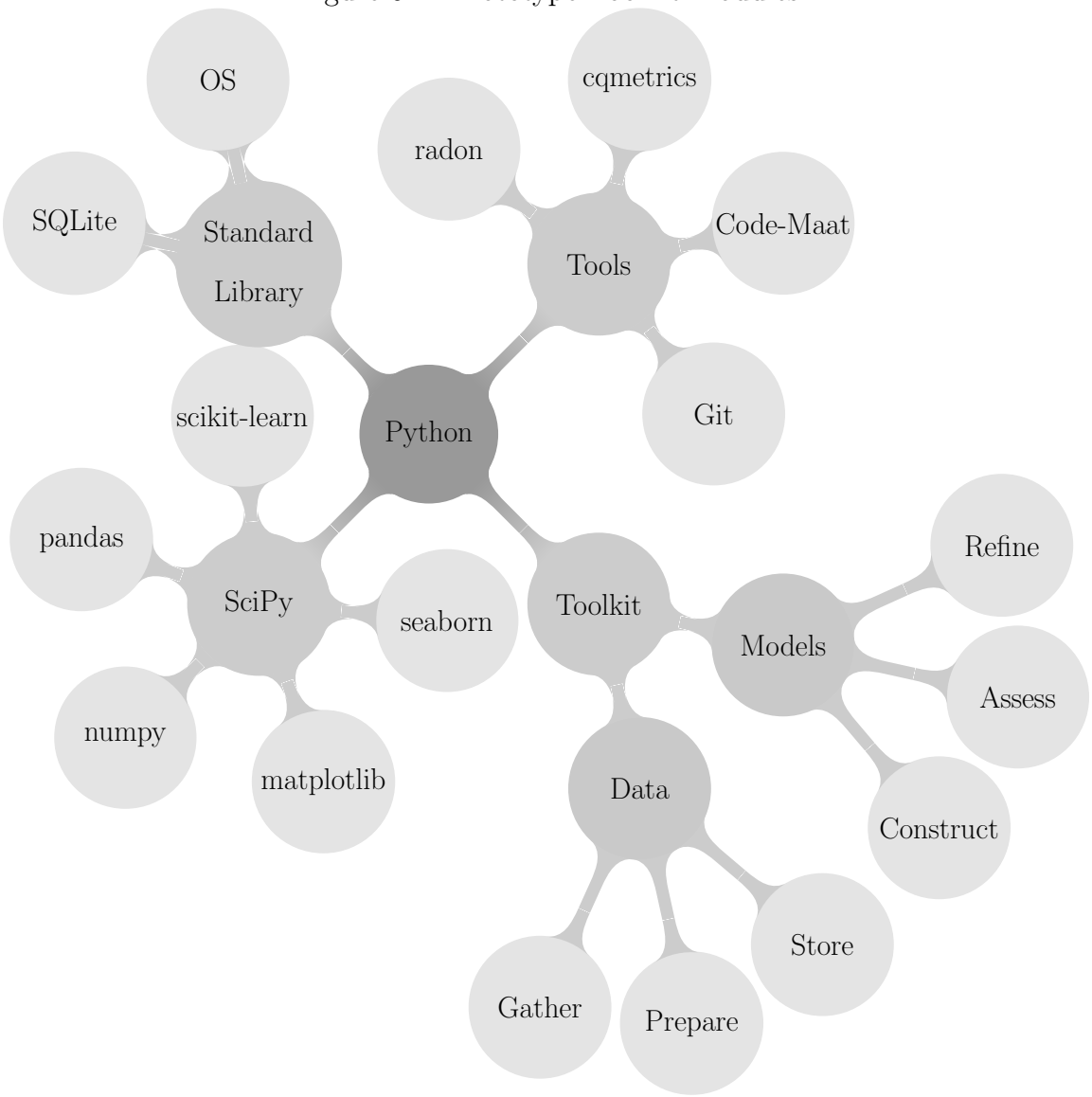
The prototype captures a minimal subset of the interfaces necessary to satisfy the design. The SCM system Git is supported as the primary data source. Language-specific measurements are provided for C and Python source code structure. Language-agnostic measurements are provided for indentation counting and SCM change metrics. The implemented modules can be adapted to support other SCM systems, source code languages and metrics tools. The prototype and all its dependencies are encapsulated in a lightweight virtual machine image, supporting repeatable usage and development on all major platforms.

We describe the components and their organization in the implemented system. Following this, we review important functions in the original toolkit modules which are used by the case study in Chapter 7.

6.1 Prototype Diagram

The prototype system is composed of original Python modules, third-party tools and libraries as shown in Figure 6.1.

Figure 6.1: Prototype Toolkit Modules



6.2 Prototype Components

The prototype leverages existing software including a programming language, execution environment, domain-specific libraries and third-party data sources. Original software was developed to instantiate and connect elements of the above, to establish and carry out research tasks. We continue with a description of each component.

Programming Language

Python

The toolkit is implemented using the Python [61] programming language. This is a multi-platform, interpreted general-purpose language designed to encourage readability and simplicity.

Python has been one of the most strongly-supported languages in the trend toward Data Science activities in software development. The SciPy community has developed many powerful libraries supporting a variety of scientific and data-oriented tasks. Many of these include exemplary documentation and community support.

Python standard library

We make use of the Python standard library for operating system functions, string handling and calls to outside processes as data sources. This helps to ensure that the toolkit is portable to a variety of platforms.

Domain-Specific Libraries

SciPy stack

We make use of some popular libraries from the scientific Python (SciPy) community. The pandas [55] library is used to represent the data under study in dataframes. In our usage, data is represented as arrays of dimension [*# of samples*][*# of features*].

Since pandas is built on top of the numpy [62] library, we apply some numpy operations to pandas dataframes. We use the scikit-learn [54] library to instantiate, train, validate and refine machine learning models.

Visualization

For visualization of scikit-learn decision tree models, we leverage its ability to output human-readable tree diagrams using graphviz [63] (a tool which formats graphs in the dot language, to be rendered as viewable images).

For visualization of other data in the system, we make use of the seaborn [64] library. Since seaborn is built using matplotlib [65], we also make direct calls to matplotlib.

SQLite

For the data storage aspects of the system, we use SQLite databases via the sqlite3 [66] library. SQLite is a lightweight database format. The entire database is stored in a single file and no additional services are required to access it. SQLite is ideal for projects having a variety of performance requirements, but it does face limits for very large-scale deployments. These are detailed in its provided user guide [67]. For the current version of the toolkit, the limitations of SQLite are unlikely to be reached. However, the current implementation of data storage can easily be extended

to support porting data sets from SQLite to another format, should the need arise. In addition to its representation of data, pandas is used to convert database tables to and from dataframes at runtime. Dataframes can act as an intermediary format between SQLite and another database, ensuring compatibility with the toolkit.

Environment Management

Docker

For ease of use, the toolkit and all its software dependencies are collected in a Docker [68] container. This is a virtual machine image which can be used to run the toolkit, view or edit its source code, and manipulate experimental data sets. The container can be used with any platform for which Docker support is available (primarily Linux and Mac OS).

Anaconda

Within the Docker container, we make use of the Anaconda [69] distribution of Python. This is used to group Python packages and their dependencies, so that compatible versions of libraries such as pandas and scikit-learn are made available to the system running the toolkit.

Data Sources

Git

In the prototype system, we gather data from projects using the Git [70] SCM system. This is our primary data source - the Git client allows us to acquire source code from the project under study and to generate revision history log files for analysis.

To add support for SCM systems other than Git, the scmData module would be updated. A module similar to gitLogInterface would be created to carry out these tasks with other SCM systems. Although not used by this implementation, libraries such as GitPython [71] can provide portable interfaces to popular SCM systems. These libraries have many of the same maintenance tradeoffs as our approach (system calls to the Git client). Changes to Git (although rare) can necessitate changes to libraries and tools which interface with it. This represents a broad issue for ESE research. Developers and organizations choose which SCM systems to use and how to use them. Researchers and automation systems must accommodate these choices in order to be effective.

Code-Maat

To measure a variety of change metrics, we use the Code-Maat tool developed by Tornhill [72]. Although Code-Maat can gather data from projects in a variety of SCM systems (Git, SVN, CVS, Mercurial), we only make use of its facilities with Git projects.

Code-Maat provides language-agnostic measures of how project files are changed between versions. This includes the number of lines added or deleted, number of contributing authors and files which are changed together. Many of the supported metrics have been previously studied in the empirical software literature, but have not been available in such a unified public-domain tool.

cqmetrics

To measure structural attributes of C source code, we use the cqmetrics tool developed by Spinellis [73]. This tool offers many traditional measures of code structure. This

includes the file size in lines, characters and comments, the number of functions, the nesting depth of control structures and the number of uses of keywords such as goto. Typically, studies interested in code structure attempt to relate such measures to target variables such as defect density, failure rate and code churn. Using the provided interface, static metrics data can be gathered uniformly about any C projects under study and used alongside other such data sources.

Radon

Using the Radon [74] tool, we measure structural attributes of Python code. This is similar to our application of the cqmetrics tool for measurement of C code. Support for this tool was added late in the development of the toolkit, to show that the system can be easily extended to support new data sources.

The toolkit supports language-agnostic change measures, and language-specific code structure metrics for C and Python projects. With all of these options available, one can carry out a study comparing similar measures of projects which use these two different languages. This represents a common goal in the empirical software literature - the search for points of reference among the many programming languages, system architectures and domains of application for software. Typically, such a study would be hampered by the complexity of data gathering and interoperation of tools. The toolkit alleviates this complexity and brings uniformity to these aspects of the research process.

Indentation

Using an original module, we measure structural attributes of indentation in files. This is a language-agnostic measure of text structure. Support for this tool was

added late in the development of the toolkit, to show that the system can be easily extended to support new data sources. This module was inspired by a tool created by Tornhill [75]. To improve performance, we have implemented the same measurements using Python.

Radon and cqmetrics can provide language-specific measures of code structure (by analyzing C and Python syntax), whereas indentation provides language-agnostic measures of structure. Indentation measures have previously been studied by Hindle [12], who used them as proxy measures of code complexity. While some programming languages (such as Python) require strict indentation of source code, others (such as C) do not. Although C does not strictly give meaning to indentation (as Python does), C programmers typically use indentation to indicate the complexity of nesting in control structures such as loops and decisions. Since this may be related to several types of code quality (e.g. defects, maintainability) it is a common target for empirical study.

By counting indentation, the toolkit is able to provide language-agnostic measures of code structure at particular points in time. When combined with other sources of data, this can allow researchers to connect indentation with other measures of software quality or to challenge the results of researchers such as Hindle [12].

By implementing indentation measures with an original module, we demonstrate how first-party data sources can be integrated into the toolkit. Researchers frequently create their own metrics and corresponding measurement tools. The design of the toolkit accommodates this approach.

Toolkit Modules

We describe the toolkit modules in terms of their responsibilities, instantiation and usage. Portions of the source code are presented to illustrate key interfaces used in the case study (Chapter 7). A full listing of the toolkit source code can be found in Appendix A.

Data

The toolkit includes several modules dedicated to data acquisition, storage and preparation. The case study makes use of two high-level functions:

- gatherTimeMetrics

  This function gathers metrics data about a group of source files over a range of time periods. Its parameters define the subset of SCM data to be measured:

  - the files to be studied
  - the range of commit dates (time periods)
  - the types of metrics to be gathered

  This function returns a Python dictionary containing the metrics data gathered (as a pandas dataframe), and the number of times measured. An example call to this function is as follows:

```
metricsData = toolkit.data.gatherTimeMetrics(
    rootDirectory, 'https://github.com/git/git',
    rootDirectory+'git/', '*/*.c */*.h *.c *.h', ['
    indent','c'], skipEvery=50)
```

- addBinnedResponseCategory

  This function converts a feature into a group of categorical variables. This is done by dividing the chosen feature into bins representing quantiles, and transforming the binned data into binary categorical features. For example, the size of source files may be grouped into four bins, which are turned into four categorical features (incdicating which source file size bin each sample belongs to).

  This function returns an updated copy of the input data set, with the chosen feature replaced by the categorical features. An example call to this function is as follows:

```
dataSetUpdated = toolkit.utilities.
    addBinnedResponseCategory(metricsData['data'], '
    netchurn', churnBinnedCategories)
```

Modelling

The toolkit includes several modules dedicated to model construction, analysis and refinement. The case study makes use of one high-level function which encapsulates all of these features into a particular workflow:

- makeAndUpdateModel

  This function performs model training, analysis and refinement given a prepared data set. Its parameters define the modelling tasks to be carried out:

  - the input data set

- the initial and refined model instances
- the response variable(s) to be used as model outputs
- the number of time series cross-validation folds

It proceeds through the following stages:

- Perform a time series cross-validation of the initial model instance on the input data set
- Select the important features from the trained initial model
- Remove unimportant features from the data set
- Perform a time series cross-validation of the refined model instance using the selected features

This function returns a Python dictionary containing the refined model and the selected features. An example call to this function is as follows:

```
from sklearn.tree import DecisionTreeClassifier
  modelInstance = DecisionTreeClassifier(max_leaf_nodes
     =8, criterion='entropy')
  modelSimpler = DecisionTreeClassifier(max_leaf_nodes=4,
       criterion='entropy')

churnModel = toolkit.refinement.makeAndUpdateModel(
    rootDirectory, metricsData['data'], 2, 'netchurn',
    modelInstance, modelSimpler, visualize=True, scoreOnly
    =False)
```

Front-End Notebook

This allows the user to carry out a workflow using the tool in an IPython notebook [50]. Results of data gathering and modelling are presented inline with executable source code blocks. The notebook can be rendered to other display formats for publication and discussion. The complete notebooks from our case study are presented in Appendix B.

Toolkit Distribution Package

This package uses Docker to encapsulate the toolkit, the front-end notebook and all software dependencies in a lightweight virtual machine. As described by Boettiger in [27], this allows one to automatically recreate the environments used to develop the toolkit and to carry out the case study. This can be useful to other researchers who may wish to use the tool for new studies, or to reproduce our study. It can also be useful to developers who wish to study or change the prototype toolkit implementation. The only requirement is for the host system to have a working Docker installation available for use.

The toolkit distribution is available at [76] and includes the following:

- Source code for the toolkit
- Data sets gathered for use in the case study
- A Dockerfile which can be used to assemble a Docker container to run the toolkit
- A readme file containing instructions for using the Docker container

In the following chapter, we describe a case study which uses the prototype toolkit to model data gathered about several software projects.

Chapter 7

Case Study

A case study was carried out to assess the implemented toolkit. We describe the design of the study, data sets gathered, results and assessments of the toolkit. Throughout this chapter, we present selected results from the IPython notebooks used in the study. The toolkit source code is presented in Appendix A and the complete IPython notebooks are presented in Appendix B.

7.1 Case Study Outline

To demonstrate the usefulness of the prototype toolkit, we designed a study with the following goals:

- Gather data about a software project implemented in C and stored using Git
- Train classification and regression models of source code size, growth and complexity
- Validate the trained models using time series cross-validation

As an additional goal, we sought to repeat the same study with several different projects as data sources. All of the studied projects were implemented in C and stored in Git repositories, but they differed in other ways. We continue by describing the projects studied and data sets gathered.

Data Sets

The study was initially carried out using data about the Git software project. Our data was gathered from the Github mirrored copy [70] of the Git SCM repository. Since the beginning of the project in April 2005, Git had received about 42,000 commits from about 1,000 contributors. In our study, we selected all commits which changed source files with the .c and .h extensions. From these, we measured every 50th commit. The resulting dataset contains 15811 samples, measuring 621 unique source files over 308 time periods.

Other projects selected for study include Vim (a text editor), OpenSSL (a security library), Apache httpd (a web server) and nginx (a web server). The projects studied had varying sample sizes and applicable time periods. The sizes of all studied data sets are summarized in Table 7.1.

Table 7.1: Case study data sets

Project	**Creation Date**	**Commits**	**Authors**	**Samples**	**Files**	**Time periods**
Git	April 2005	42615	1004	15811	621	308
Vim	June 2004	5896	1[1]	2684	193	105
OpenSSL	December 1998	16393	173	14042	1351	180
httpd	June 1996	28525	27	10054	1866	253
nginx	August 2002	5677	28	3457	481	88

Analysis Steps

For each of the projects under study, we performed the same modelling and analysis steps. We began by carrying out our analysis of the Git dataset. The front-end IPython notebook [50] used for this was re-applied using each of the other data sets. We leveraged the automatic workflow steps in the toolkit and the uniformity of IPython notebook presentation to create a group of comparable studies.

The steps performed within each study were as follows:

- Gather change and static metrics data about .c and .h files from every 50th commit in the repository.
- Carry out the following for each model to be created:
  - Train the model. Cross-validate it in terms of its accuracy measures.
  - Refine the trained model to be simpler by removing unimportant features. Cross-validate and assess the refined model.
  - Analyse the data relationships represented by the trained model.
  - Remove features from the dataset which may be intrinsically related. Repeat the above modelling steps.

The following models were trained:

(1) Binary classification tree for net churn (above and below the mean).

(2) Multiple classification tree for net churn (with five classes: low, medium-low, medium-high, high, very high).

[1] At the time of writing, Bram Moolenaar is the only user to commit to the Vim repository. The original authors of many changes are listed within commit messages in the Vim repository.

(3) Regression trees for file size in lines and maximum cyclomatic complexity.

(4) Binary and multiple classification trees for mean indentation level. Random forest classifier for mean indentation level.

7.2 Git Study Results

We begin the analysis by training Model (1) on the Git dataset. This model makes a binary classification of net churn above and below the mean value. The toolkit trains the model on our dataset, performs cross-validation with two folds (each set covering half of the measured time periods) and outputs measures of its prediction quality.

We make the following observations about our model and its assessment (shown in Figure 7.2):

- The model has high Precision, Recall and F1-Score. Net churn above/below the mean is classified well by this model.
- ROC area under curve is high. There is little compromise between true positive rate and false positive rate.
- The model says that the features influencing net churn are (strongest to weakest)
  - Number of lines added
  - Number of lines deleted
- The 'net churn below the mean' class is over-represented in our data (three times as many samples as for the other class). However, the model still performs

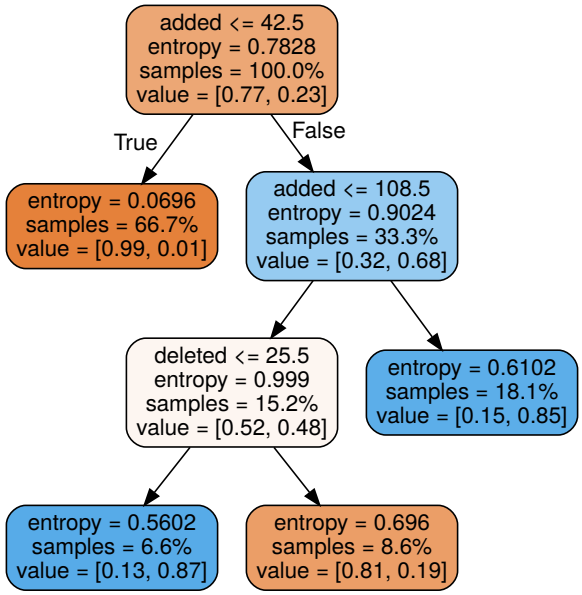
well without any steps taken to address class imbalance (e.g. undersampling, oversampling).

Figure 7.2: Model (1), assessment

```
Model.score: 0.959135
             precision    recall  f1-score   support
          0       0.97      0.98      0.97      6110
          1       0.92      0.89      0.91      1794
avg / total       0.96      0.96      0.96      7904
roc_auc_score: 0.935571
      name  importance
3    added    0.847304
4  deleted    0.152696
```

At this stage, the toolkit refines the model. Originally, the model was restricted to have a maximum of eight leaf nodes. Refinement restricts the model to four leaf nodes, and removes unimportant features from the dataset (leaving only lines added and lines deleted). The refined model is visualized by the toolkit, as shown in Figure 7.3. Leaf nodes represent classified groups of samples and intermediate nodes represent decisions. The colour of each node indicates its likely classification (orange for net churn below the mean, blue for net churn above the mean). Darker nodes have more confident classifications (given by *value*).

Figure 7.3: Decision Tree visualization



We observe the following about the visualized model:

- 67% of files have less than 43 lines added.
- These files have net churn far below the mean, and are stable.
- The files with net churn greater than the mean had more than 108 lines added.

We continue by training and analysing Model (2). We divide the net churn feature into five bins representing quantiles, and transform the binned data into five binary categorical features. We train a decision tree classifier to predict these five classes of net churn.

We make the following observations about our model and its assessment (shown in Figure 7.4):

- The vast majority of samples exhibit a medium-to-high amount of code churn.

- The model is able to classify four of the five levels of churn accurately. The model does not correctly classify any of the samples with the highest churn.

A simple analysis of the classification metrics would not reveal its inability to categorize the files with the highest churn. A refined version of this decision tree (with four leaves rather than eight) is similarly unable to predict this class. To improve this, a different type of model may need to be trained. One could also apply techniques to address class imbalance, such as oversampling or undersampling of the input data.

Figure 7.4: Model (2), assessment

```
Response variable was ['churnLow', 'churnMedium', 'churnHigh',
            'churnHigher', 'churnHighest']
Model.score: 0.996457
              precision    recall  f1-score   support
          0       1.00      1.00      1.00         9
          1       0.96      0.81      0.88        97
          2       1.00      1.00      1.00      7762
          3       0.86      0.94      0.90        32
          4       0.00      0.00      0.00         4
avg / total       1.00      1.00      1.00      7904
roc_auc_score: 0.860955
            name  importance
4        deleted    0.548503
3          added    0.383638
0     age-months    0.034864
57  indent_mean    0.032995
```

We continue by training and analysing Model (3). We begin by training a decision tree regression model of file size in lines.

We make the following observations about our model and its assessment (shown in Figure 7.5).

- We see that the number of indented lines is a good predictor of the number of lines.
- However, these two features are likely to be correlated in any large dataset with typical C code. We compute the Spearman rank correlation [77] between these features as 0.87 for our dataset. There is a strong monotone relationship between the number of lines and the number of indented lines.
- We proceed by removing from our dataset features which, although they accurately predict the size of files, likely do so because they are intrinsically related to it. We remove the following features: *indent_lines, nchar, nstatement, nidentifier, nfunction, unique_nidentifier.*

Figure 7.5: Model (3), file size

```
Model.score: 0.963244
name  importance
2          indent_lines     0.991665
1    unique_nidentifier     0.004586
3   entity_cache-tree.c     0.003749
```

The revised model (shown in Figure 7.6) loses a significant amount of accuracy. Without the metrics which most strongly characterized the large files, it must use *ninternal* (the number of variables with static linkage) as the discriminating feature.

Figure 7.6: Model (3), file size (revised)

```
Model.score: 0.725020
        name  importance
0  ninternal    0.802897
2    nstruct    0.102610
3   ncomment    0.094493
```

It is not illuminating to say that the larger files have more indented lines, more characters or more functions. However, it is somewhat interesting that the large files have more data structure declarations and static variables than the smaller files. Such a statement could lead to a characterization of the change-prone source files in this project.

We continue by training a regression model of the maximum cyclomatic complexity in files.

Figure 7.7: Model (3), cyclomatic complexity

```
Model.score: 0.579328
                     name  importance
3             halstead_max    0.865349
0               nstatement    0.088993
1  statement_nesting_mean    0.045658
```

We make the following observations about our model and its assessment (shown in Figure 7.7):

- The model has poor performance in general (with an R^2 error score of (1 - 0.57))
- We see that cyclomatic complexity is strongly predicted by Halstead complexity and by the level of statement nesting. Of course, these metrics are related by definition.

- We proceed by removing the following features: *halstead_sd, nidentifier, halstead_mean, halstead_min, cyclomatic_sd, cyclomatic_mean, halstead_max, nstatement, statement_nesting_mean*

The revised regression model is even less accurate. It uses *ninternal* as the discriminating feature.

```
Model.score: 0.474307
          name  importance
0    ninternal    0.630822
4  indent_mean    0.369178
```

Perhaps we cannot accurately regress on cyclomatic complexity, but we may be able to classify it into coarse-grained categories. To test this, we train the previously-used decision tree classifier on maximum cyclomatic complexity above and below the mean. We continue using the dataset with strongly-linked features removed. The results from this model are shown in Figure 7.8.

Figure 7.8: Model (3), cyclomatic complexity classification

```
Model.score: 0.841979
             precision    recall  f1-score   support
          0       0.94      0.83      0.88      5548
          1       0.68      0.88      0.77      2356
avg / total       0.86      0.84      0.85      7904
roc_auc_score: 0.853126
          name  importance
4  indent_mean    0.749675
0    ninternal    0.250325
```

Although this model has difficulty predicting the minority class, it performs better than the regression models. Interestingly, the mean level of indentation is a more important feature than *ninternal* had been previously. The measured source files may have cyclomatic complexity characterized by their indentation. This is not difficult to accept, given the typical practices of indentation used when programming

C control structures (e.g. for loops, if statements).

Using a similar approach to the above, we create a binary classification tree for Model (4). This models the mean indentation level above and below the mean. The results from this model are shown in Figure 7.9.

Figure 7.9: Model (4), binary classification

```
Model.score: 0.908148
             precision    recall  f1-score   support
          0       0.82      0.96      0.89      2917
          1       0.98      0.88      0.92      4987
avg / total       0.92      0.91      0.91      7904
roc_auc_score: 0.919242
                   name  importance
5          halstead_max    0.602991
3  statement_nesting_sd    0.302799
1            nstatement    0.094210
```

As in the analysis of Model (2), we create quantile bins of mean indentation level and train a multi-classification tree on the data (shown in Figure 7.10).

Figure 7.10: Model (4), multiple classification

```
Model.score: 0.712171
             precision    recall  f1-score   support
          0       0.80      0.78      0.79      1849
          1       0.54      0.74      0.62      2285
          2       0.85      0.69      0.76      3622
          3       0.00      0.00      0.00       148
avg / total       0.73      0.71      0.71      7904
roc_auc_score: 0.723178
                   name  importance
3         cyclomatic_sd    0.819947
2  statement_nesting_sd    0.180053
```

This model suffers from class imbalance. With only eight leaves, it is unable to predict the files with the most indentation. This is a common issue with decision trees being exposed by our dataset. It is not a property of the studied codebase or

of C source code. In this case, we may be able to use an ensemble model to make up for the loss of accuracy in the minority class. However, we do so at the expense of model interpretation.

To illustrate this, we train a *random forest*. A group of 500 decision trees is trained with the first decision in each chosen at random. Each tree classifies each sample, and a majority voting scheme decides the ensemble output. This has the effect of creating more trees. Some may resemble the above tree (which accurately modelled most of our data). Others may be highly inaccurate, except for small subsets of the data.

Forest models can be difficult to tune and to interpret. We omit visualization of the many trees. For automated classification tasks used in a production environment (as opposed to a research environment), forests may be valuable despite their reduced interpretability. An advantage of random forests is that the model training step is 'embarassingly parallel', allowing the use of all available CPU cores to train the component decision trees.

The resulting forest (shown in Figure 7.11) uses many features (most of which have low importance). Many of these are named entities - the trees are individually modelling the files in the codebase. This showcases a possible threat to the use of forest models with our dataset - there may be a tendency to model the one-hot encoded entities.

We also note that the minority class is predicted with high Precision, but low Recall. Within the high-indentation files we predict, they are predicted correctly. However, most of the high-indentation files are missed even by this model. The F1-Score (the harmonic mean of Precision and Recall) is therefore low.

Figure 7.11: Model (4), random forest

```
Model.score: 0.899038
precision    recall  f1-score   support
0        0.95      0.95      0.95      1849
1        0.87      0.86      0.87      2285
2        0.92      0.92      0.92      3622
3        1.00      0.21      0.35       148
avg / total       0.91      0.90      0.90      7904
roc_auc_score: 0.851069
name    importance
17          statement_nesting_mean  5.533735e-02
54                   cyclomatic_sd  4.705789e-02
...
135         entity_builtin-merge.c  4.138996e-08
298                entity_column.h  3.683571e-08
373          entity_ewah_ewah_io.c  3.655622e-08
557              entity_sigchain.h  3.353482e-08
...
```

7.3 Results from other data sets

By leveraging the automation features of the toolkit, the above analyses were repeated for other data sets. We summarize the results by comparison with the Git study.

Vim

Results from this study are shown in 7.12. Multi-classification of net churn quantiles was unsuccessful for minority classes with the most churn. Lines deleted were a stronger discriminator than lines added.

Regression on file size was accurate. The most important feature was the size of comments (in characters). The largest files in the Vim codebase may be dominated by comments (rather than lines of executable code).

Figure 7.12: Vim Results

```
Response variable was
['churnLow', 'churnMedium', 'churnHigh',
 'churnHigher', 'churnHighest']
Model.score: 0.976883
             precision    recall  f1-score   support
          0       0.25      1.00      0.40         3
          1       1.00      1.00      1.00      1286
          2       0.61      0.74      0.67        34
          3       0.00      0.00      0.00        15
          4       0.00      0.00      0.00         3
avg / total       0.97      0.98      0.97      1341
roc_auc_score: 0.760412
      name  importance
4  deleted    0.704087
3    added    0.295913

Response variable was nline
Model.score: 0.831832
            name  importance
2  ncomment_char    0.889481
4  indent_median    0.082914
0          ngoto    0.027605
```

OpenSSL

Results from this study are shown in 7.13. Multi-classification of net churn quantiles was accurate for the three identifiable classes of churn. The data could not be split into five classes by the pandas function used in the toolkit. The age of files (the time since they were last changed) was a weak predictor of churn. Perhaps a particular time period in the data set involved large changes to some files which were then stabilized. Further inquiry would be required to investigate this result.

Binary classification of indentation level had acceptable performance, but struggled to predict the files with indentation level above the mean. Multiple classification of indentation level struggled with predicting the minority classes which had the most indentation.

Figure 7.13: OpenSSL Results

```
Response variable was
['churnLow', 'churnMedium', 'churnHigh',
 'churnHigher', 'churnHighest']
Model.score: 0.999003
             precision    recall  f1-score   support
          0       0.00      0.00      0.00         0
          1       0.92      1.00      0.96        12
          2       1.00      1.00      1.00      6973
          3       0.97      0.86      0.91        35
          4       0.00      0.00      0.00         0
avg / total       1.00      1.00      1.00      7020

                  name  importance
3                added    0.564922
4              deleted    0.382906
0           age-months    0.027663
40  nfun_cpp_directive    0.024508

Response variable was indent_mean
Model.score: 0.807265
             precision    recall  f1-score   support
          0       0.77      0.94      0.84      3895
          1       0.89      0.65      0.75      3125
avg / total       0.82      0.81      0.80      7020
roc_auc_score: 0.791396
                     name  importance
5             nidentifier    0.610091
1  statement_nesting_mean    0.243350
3                   ngoto    0.146560

Response variable was ['iLow', 'iMedium', 'iHigh', 'iVeryHigh']
Model.score: 0.652991
             precision    recall  f1-score   support
          0       0.82      0.51      0.63      2653
          1       0.58      0.87      0.70      3219
          2       0.75      0.40      0.52      1102
          3       0.00      0.00      0.00        46
avg / total       0.70      0.65      0.64      7020
roc_auc_score: 0.644799
```

httpd

Results from this study are shown in 7.14. Classification of net churn above and below the mean exhibited high Precision, but low Recall. A significant discriminating feature was *soc*, the sum of coupled changes. This is a measure of how many changes to a file were coupled to simultaneous changes in other files. Code churn in the httpd codebase may be strongly predicted by certain groups of files which change together (because they share responsibilities). Further inquiry would be needed to investigate this hypothesis.

Multi-class modelling of the same data revealed a pathological data set - the vast majority of samples fall into just one of the four identifiable classes of churn. It is possible that both the OpenSSL and httpd data sets exhibited this phenomenon, but there may also be an issue with how the toolkit transforms data into multi-class categorical variables. Further inquiry could lead to changes in the toolkit source code, followed by re-application of the study to these data sets.

Models of indentation level used as important features the number of statements in files, as well as McCabe and Halstead [13] complexity. This is similar to the results of Hindle [12] which connected indentation with measures of syntactic complexity.

Figure 7.14: httpd Results

```
Response variable was netchurn
Model.score: 0.923199
             precision    recall  f1-score   support
          0       0.92      1.00      0.96      4383
          1       0.94      0.43      0.59       643
avg / total       0.92      0.92      0.91      5026
roc_auc_score: 0.711124
      name  importance
0    added    0.494887
2      soc    0.370823
1  deleted    0.134291

Response variable was
['churnLow', 'churnMedium', 'churnHigh',
 'churnHigher', 'churnHighest']
Model.score: 0.998607
             precision    recall  f1-score   support
          0       0.00      0.00      0.00         0
          1       0.00      0.00      0.00         1
          2       1.00      1.00      1.00      5019
          3       0.00      0.00      0.00         5
          4       0.00      0.00      0.00         1
avg / total       1.00      1.00      1.00      5026

      name  importance
1  deleted    0.596914
0    added    0.403086

Response variable was indent_mean
Model.score: 0.822125
             precision    recall  f1-score   support
          0       0.94      0.69      0.80      2530
          1       0.75      0.96      0.84      2496
avg / total       0.85      0.82      0.82      5026
roc_auc_score: 0.823046
             name  importance
1      nstatement    0.653101
3    halstead_max    0.265817
4  cyclomatic_max    0.081082
```

nginx

Results from this study are shown in 7.15. Models of indentation level were similar to those in the httpd study. For the studies of httpd and nginx, the number of statements in files was a stronger predictor of indentation than the complexity of functions. This may be a misleading statement, since it is likely that *only* statements are indented in the typical style of writing C programs. Function headers, global data and file-level comments are likely to be unindented. A refinement of these models without using the number of statements would likely rely on complexity measures for prediction. However, its performance could suffer as a result. Further analysis could be done to assess this.

Figure 7.15: nginx Results

```
Response variable was indent_mean
Model.score: 0.883034
             precision    recall  f1-score   support
          0       0.83      0.98      0.90       915
          1       0.97      0.78      0.86       812
avg / total       0.90      0.88      0.88      1727
roc_auc_score: 0.877002
                    name  importance
0             nstatement    0.745331
6      cyclomatic_median    0.158924
2  statement_nesting_sd     0.095745
Response variable was ['iLow', 'iMedium', 'iHigh', 'iVeryHigh']
Model.score: 0.842501
             precision    recall  f1-score   support
          0       0.91      0.94      0.93       817
          1       0.76      0.85      0.80       566
          2       0.82      0.61      0.70       297
          3       0.87      0.55      0.68        47
avg / total       0.84      0.84      0.84      1727
roc_auc_score: 0.838030
                      name  importance
15               nstatement    0.550109
17  statement_nesting_mean    0.348854
8                     nchar    0.045749
20    statement_nesting_sd    0.027776
21                ninternal    0.027512
```

7.4 Discussion

Through the case study, we have encountered several typical issues with machine learning models of software metrics data.

- Precision, Recall and overall model score can be deceptive measures. They may hide unacceptable weaknesses in multiple classification models, or in models of class-imbalanced data sets.
- Regression may be inappropriate for some target features. Classification into coarse-grained categories may be simpler and more effective.
- Random forests can improve performance over single decision trees, at the expense of model complexity and interpretability. However, training may also produce overfit forests. In the case of Model (4), the trained forest modelled the individual source files rather than their static or change metrics. Further tuning would be required to apply random forests to this problem.
- Common observations can be made about models of general software metrics (such as size, complexity and churn) across the studied data sets.
- Pathological qualities of the data sets can greatly influence model training and prediction. Examples include strongly-coupled files and periods of high churn followed by stability.
- Peculiar results that occur repeatedly may be a sign of issues with the toolkit or the underlying tools. These can be targeted for further development, to improve the results of future studies.

Our study demonstrates that simple applications of the toolkit can automate large-scale modelling tasks. The effort of researchers can therefore be focused on careful analysis of model training and validation results. Repeatedly applying the toolkit to a variety of data sets can also reveal issues with its measurements or the tools which it relies upon. We conclude the case study by discussing these and other issues of experimentation.

7.5 Threats to Validity

Our case study showcased several areas in which the toolkit addresses experimental validity. We give an overview of how threats to validity can be mitigated by experimental design and by improvements to the toolkit itself.

Content Validity

We did not gather data about every revision in the studied Git repositories. However, our gathered data encompasses the full lifetime of the studied projects (from their creation until the present day), with measurements collected after every 50th commit. In [24], it was noted that the tools used were unable to gather data about older SCM revisions. Therefore, only the applicable newer revisions were selected for study. By limiting our study to simple, general software metrics, we avoided this issue.

Construct Validity

The cqmetrics [73] tool provides many static code metrics which are closely related. We have attempted to mitigate this by removing related features from our datasets, and gradually revising models. For example, we found that the number of indented lines was a strong predictor of the number of lines, and therefore removed the number of indented lines before re-modelling.

Similar issues have arisen in many studies of software metrics. As we have illustrated, Halstead and McCabe complexity often correlate with their component measures (degree of statement nesting and number of statements). By iteratively refining data and models, the toolkit allows researchers to study the effect of such correlation and resolve the experimental issues related to it.

Internal Validity

Our experiment attempts to study large projects over long time periods. However, we have limited our results by selectively measuring .c and .h files at particular time intervals. We measured both static metrics and change metrics in order to study the relationships between these distinct measurement types. We measured language-agnostic structure (indentation) in order to limit the bias toward measures which depend upon C syntax.

Additionally, our machine learning models perform biased choices. There is a well-known tradeoff between bias and variance [78] in classification models. In order to limit the variance among the constructed trees (and forests), we have limited the size of trees by the number of leaf nodes. This allows us to adjust other parameters of model construction and observe the effect on prediction bias.

External Validity

Although our study covers source code from a variety of applications, they are all implemented using C. The study could be extended to measure source code written in other languages and compare with the results about C projects. As Table 7.1 shows, the studied projects vary by the number of commits, number of authors, number of files and time periods. An important advantage of the toolkit is that it can accommodate data sets of varying scope.

Since all of the studied projects are open-source applications with worldwide authorship, they may not exhibit the characteristics of a particular programming style or development process. In order to study those characteristics, one may need to constrain their study to data from a particular organization. This was the case in [19], which used only internal data from Microsoft.

We continue by identifying several goals for future development of the toolkit.

Chapter 8

Future Work

The project concludes having developed and used the research toolkit. The prototype automates a defined workflow, with many interfaces open to extension. Future development of the toolkit could pursue several different goals. We describe these as follows.

New Data Sources

The toolkit includes support for the measurement of change metrics and static code metrics. It also includes a module for language-agnostic measurement of indentation in source code files. Together, these measurements formed the feature set gathered in the case study. Throughout the many branches of ESE research, a variety of other measurements have been proposed. An important area for extension of the toolkit is the integration of these other measurements.

Currently-supported static code metrics include size of files, nesting depth or complexity, and the number of keyword uses. A related family of measurements is

concerned with the connections between entities in object-oriented programs. For example, the metrics proposed by Chidamber and Kemerer in [79] have been widely used to characterize architectural notions such as coupling and cohesion between Java classes. By supporting such metrics in the toolkit, one would allow the automatic creation of models which connect the object-oriented metrics to other measures of static structure and change. This would allow one to characterize, among other things, the classes which require the most frequent changes and the coupling of modules which exhibit high complexity. Such models could support managerial decisions about architectural design and refactoring tasks.

The measures used throughout our work consider only the form of source code files, and changes to their form over time. They do not consider compilation or execution of the studied software projects. Fortunately, many tools are available to measure the dynamic characteristics of software, some with the ability to attribute measurements to the corresponding source files. One such tool is the Valgrind [80] suite, which can be used to audit C programs for common memory safety mistakes, profile CPU usage and identify threading issues. One could apply the modelling approach used by the toolkit to the measurements provided by tools such as Valgrind and gprof [81]. For example, it may be possible to create models which characterize memory leaks using static code attributes or change metrics. A similar study in [40] led to the creation of models which proactively rejuvenate software systems before they would fail due to resource consumption. One could use the toolkit to extend these results by characterizing which attributes of source files contribute strongly to resource consumption.

A common target of ESE research is defect prediction. Typically, this is modelled using failure data gathered during testing. This may include reports from automation systems or issue trackers. Although these measurements have often been used to create software reliability models, their collection requires a significant time investment. Studies such as [3] and [23] used ad hoc methods to collect and process failure data. Alternatively, studies such as [24] attempted to construct proxy measures of software defects. They considered software changes to be indicative of defects being fixed. Since not all changes are intended to fix defects, this became a source of error in modelling. The distinction between enhancements and fixes poses a significant challenge to such studies. Although systems exist to record the intent behind SCM changes, such systems are not used consistently by developers. The study in [3] noted the lack of meaningful messages in SCM commit data. Failure reports from automated test systems and customer installations exhibit a similar fundamental difficulty - something must record and map the reports to software artifacts before modelling can be carried out. The toolkit is able to automatically measure general software attributes, but the data required to construct reliability models is not generally available or measurable. The development of a generalized, automated approach to failure data collection would be valuable. This would allow failure models to be created without human intervention, just as we have created models of structure and change. It would also allow failure models to be connected with the other metrics available in the toolkit.

Support for these and other measurements would significantly increase the utility of the toolkit. In addition to variety in the types of measurements, one could also enhance the ability of the toolkit to aggregate its findings.

Aggregation Mechanism

Software development frequently creates hierarchies of components. Lines of code are grouped into functions, functions are grouped into classes, classes are grouped into modules and modules are grouped into packages. Although the terminology may vary between projects, there is a general notion of aggregation in how developers organize their work. This was explored by Posnett et al. in [25]. As they explained, empirical measurement of software can include statements about aggregated components which may not apply to disaggregated components, and vice versa. Statements about the separation of responsibilities and divisions between components can provide empirical validation for software development concepts such as *information hiding* [82]. Support for automatic grouping of measurements would allow the toolkit to make ecological inferences as described by Posnett, to model the (dis)similarity of measurements at different levels of aggregation. Unfortunately, the grouping of software measurements is not well understood. Briand et al. have attempted in [83] to develop a rigorous theory of software measurement, with consideration for modules and their relationships. Further research into this area may lead to the development of techniques for software data aggregation, which could extend the capabilities of the toolkit.

Continuous Integration

This project is fundamentally concerned with automation. One wishes to create software systems which automate data collection and modelling tasks, so that the effort of researchers can be directed toward analysis and inquisition. For industrial

software developers, automation often serves another purpose. Continuous integration systems such as Jenkins [84] have been used to automatically schedule and carry out incidental tasks of software development such as compilation, testing and deployment. In [39], the authors outlined a vision for software quality modelling based on this. A system which periodically rebuilds and tests software under development could also periodically gather metrics data and train models. In fact, the current prototype toolkit could be operated by a continuous integration system for C projects using Git.

Interoperation

Although the current toolkit could be added to a continuous integration schedule, its interoperability with data science tools could also be improved. Currently, the gathered data is stored as pandas [55] dataframes in SQLite [67] databases. For the toolkit to work with other systems, support must be added for exporting the database contents to external applications. Trained models from the toolkit are Python objects in the format used by scikit-learn [54]. For these to be exported and generalized, one could use an interoperation format such as the Predictive Model Markup Language [85].

Progress in the above areas would allow the toolkit to be used more effectively by researchers and developers. Collecting a wide variety of metrics at many levels of aggregation could allow researchers to unify and extend results from the majority of ESE studies. By treating the toolkit as a stage in the software development workflow and providing results in a standard format, academic and industrial researchers could create an evolving shared data source.

Chapter 9

Conclusion

This thesis described the development of a toolkit for the measurement and exploratory modelling of software metrics data. We began with a review of literature about empirical software engineering, software metrics and research challenges addressed by contemporary data science techniques.

Using a prototype-driven approach, we explored the domains of software measurement and machine learning to create a prototype toolkit. The prototype was used to carry out a case study characterizing the size, complexity and growth of modules in C programs. Our analysis of the case study results identified and explored several typical issues with machine learning models of software metrics data.

We identified several goals for future development of the toolkit. Achieving these would allow the toolkit to be applied by a wider audience of industrial software engineers and academic researchers.

This work is one of many which seek an empirical basis for software engineering. We look forward to the future of our field as data-driven approaches continue to mature.

Appendix A

Toolkit Source Code

toolkit.py

This is the top-level module which can be imported by Python environments to access the toolkit modules. It imports the modules which comprise the toolkit.

```
import dataGathering as data

import modelRefinement as refinement
import modelAnalysis as analysis

import dataUtilities as utilities
import presentation
import storage
```

scmData.py

This imports the modules which can be used to gather SCM data.

```
import gitLogInterface as gitlog
```

changeMetrics.py

This imports the modules which can be used to gather change metrics data.

```
import codeMaatInterface as codemaat
```

staticMetrics.py

This imports the modules which can be used to gather static code metrics data.

```
import cqmetricsInterface as cqmetrics
import radonInterface as radon
import indentationInterface as indentation

def gatherStaticMetrics(language, time, changeMetrics, sourceFilesDirectory,
    storageConnection):
  languageSpecific = None
  languageAgnostic = None
  if 'c' in language:
    languageSpecific = cqmetrics.codeMetricsTable(time, changeMetrics,
    sourceFilesDirectory, storageConnection)
  elif 'python' in language:
    languageSpecific = radon.codeMetricsTable(time, changeMetrics,
    sourceFilesDirectory, storageConnection)
  if 'indent' in language:
    languageAgnostic = indentation.indentMetricsTable(time, changeMetrics,
    sourceFilesDirectory, storageConnection)

  if languageSpecific is not None and languageAgnostic is not None:
    staticMetrics = languageSpecific.merge(languageAgnostic, on='entity')
  elif languageSpecific is not None and languageAgnostic is None:
    staticMetrics = languageSpecific
  elif languageAgnostic is not None and languageSpecific is None:
    staticMetrics = languageAgnostic
  else:
    return None
  return staticMetrics
```

gitLogInterface.py

This module provides routines for gathering data from the Git SCM system.

```
import os
import subprocess
import pandas as pd
from StringIO import StringIO

def isGitRepo(path):
    return subprocess.call(['git', '-C', path, 'status'], stderr=subprocess.STDOUT,
    stdout = open(os.devnull, 'w')) == 0

def cloneGitRepo(baseDirectory,repositoryURL):
  os.system("cd %s; git clone %s" % (baseDirectory, repositoryURL))

def switchToRevision(baseDirectory,revisionHash):
  os.system("cd %s; git checkout %s" % (baseDirectory, revisionHash))

def makeGitLog(sourceDirectory,filesToInspect,lastTimestamp,previousTimestamp,
    storageConnection):
  import storage
  tableName = 'gitLog_'+lastTimestamp+'_'+previousTimestamp

  if storage.tableExists(tableName, storageConnection):
    return

  command = "cd %s; git log %s...%s --pretty=format:'[%%h] %%aN %%ad %%s' --date=
    short --date-order --numstat -- %s" % (sourceDirectory, lastTimestamp,
    previousTimestamp, filesToInspect)
  gitlog = subprocess.check_output(command, shell=True)

  storage.writeFile(tableName, storageConnection, gitlog)

def makeCommitDateMapping(sourceDirectory,filesToInspect,storageConnection,branchname
    ='master'):
  import storage
```

```
  tableName = branchname + '_commitDates'
  if storage.tableExists(tableName, storageConnection):
    return storage.readTable(tableName, storageConnection)

  switchToRevision(sourceDirectory,branchname)

  command = "cd %s; git log --pretty=format:'%%h,%%ad' --date=short -- %s" % (
    sourceDirectory, filesToInspect)
  try:
    dates = pd.read_csv(StringIO(subprocess.check_output(command, shell=True)), names
    =['sha','date'],dtype={'sha': str, 'date': str})
    storage.writeTable(tableName, storageConnection, dates)
  except subprocess.CalledProcessError:
      dates = "no commit date mapping available"

  return dates
```

codeMaatInterface.py

This module provides routines for gathering data from Code-Maat.

```
import os
import subprocess
import pandas as pd
from StringIO import StringIO

def runAnalysis(logFile,logFileType,analysisType):
  codeMaatPath =  os.path.dirname(os.path.abspath(__file__)) + "/../code-maat"
  command = "cd " + codeMaatPath + "; lein run -l %s -c %s -a %s" % (logFile,
    logFileType, analysisType)
  return subprocess.check_output(command, shell=True)

def runAnalysisList(logFile,logFileType,analysisList):
  results = []
  for analysisType in analysisList:
    results.append(StringIO(runAnalysis(logFile,logFileType,analysisType)))
  return results
```

```
def changeMetricsTable(lastTime, previousTime, logFileType, storageConnection):
  import storage

  tableNameChange = lastTime + '_changeMetrics'
  if storage.tableExists(tableNameChange, storageConnection):
    return storage.readTable(tableNameChange, storageConnection)

  tableNameLog = 'gitLog_'+lastTime+'_'+previousTime
  scmLogPath = storage.readFile(tableNameLog, storageConnection, toTemporaryFile=True
    )

  results = runAnalysisList(scmLogPath,logFileType,entityAnalysisTypes)

  changeMetrics = pd.read_csv(results[0])
  for result in results[1:]:
    changeMetrics = changeMetrics.merge(pd.read_csv(result), on='entity')

  changeMetrics.drop('n-revs_y', axis=1, inplace=True)
  changeMetrics = changeMetrics.rename(columns = {'n-revs_x':'n-revs'})

  storage.writeTable(tableNameChange, storageConnection, changeMetrics)

  return changeMetrics

entityAnalysisTypes = [ 'age', 'authors', 'entity-churn', 'fragmentation', 'revisions
    ', 'soc' ]
```

cqMetricsInterface.py

This module provides routines for gathering data from cqmetrics.

```
import os
import subprocess
import pandas as pd
import numpy as np
from StringIO import StringIO
```

```
cqMetricsPath = os.path.dirname(os.path.abspath(__file__)) + "/../cqmetrics/"

def runAnalysisOnFile(fileToMeasure):
  command = cqMetricsPath + 'qmcalc -a %s' % (fileToMeasure)
  try:
      output = pd.read_csv(StringIO(subprocess.check_output(command, shell=True)),
    delimiter = '\t', names=featureHeader)
      return output
  except subprocess.CalledProcessError:
      output = pd.DataFrame(np.nan, index=[0], columns=featureHeader)
      output['entity'] = fileToMeasure
      return output

def dropUnusedCMetrics(metricsTable):
  for name in metricsTable.columns:
    if name not in usedFeatureNames:
      metricsTable.drop(name, axis=1, inplace=True)

  return metricsTable

def codeMetricsTable(nameLabel, dataFrame, sourceFilesDirectory, storageConnection):
  import storage
  import scmData as scm
  import dataUtilities

  tableName = nameLabel + '_codeMetrics'
  if storage.tableExists(tableName, storageConnection):
    return storage.readTable(tableName, storageConnection)

  scm.gitlog.switchToRevision(sourceFilesDirectory,nameLabel)

  metricsData = []

  for entityName in dataFrame['entity']:
    metricsData.append(runAnalysisOnFile(sourceFilesDirectory+entityName))
```

```
  if not metricsData:
    return None

  metricsData = pd.concat(metricsData)

  metricsData = dropUnusedCMetrics(metricsData)

  metricsData = dataUtilities.formatEntityNames(metricsData, sourceFilesDirectory)

  storage.writeTable(tableName, storageConnection, metricsData)

  return metricsData

usedFeatureNames = ['entity','nchar','nline','line_length_mean', 'line_length_median
    ', 'line_length_max','line_length_sd', 'nfunction', 'nstatement','
    statement_nesting_min', 'statement_nesting_mean','statement_nesting_median', '
    statement_nesting_max','statement_nesting_sd', 'ninternal', 'nconst', 'nenum', '
    ngoto','ninline', 'nnoalias', 'nregister', 'nrestrict', 'nsigned','nstruct', '
    nunion', 'nunsigned', 'nvoid', 'nvolatile','ntypedef', 'ncomment', 'ncomment_char
    ', 'ncpp_directive','ncpp_include', 'ncpp_conditional', 'nfun_cpp_directive','
    nfun_cpp_conditional', 'nfunction2', 'halstead_min','halstead_mean', '
    halstead_median', 'halstead_max', 'halstead_sd','nfunction3', 'cyclomatic_min', '
    cyclomatic_mean','cyclomatic_median', 'cyclomatic_max', 'cyclomatic_sd','
    nidentifier', 'unique_nidentifier']
```

```
featureHeader = ['nchar','nline','line_length_min','line_length_mean','
    line_length_median','line_length_max','line_length_sd','nfunction','nstatement','
    statement_nesting_min','statement_nesting_mean','statement_nesting_median','
    statement_nesting_max','statement_nesting_sd','ninternal','nconst','nenum','ngoto
    ','ninline','nnoalias','nregister','nrestrict','nsigned','nstruct','nunion','
    nunsigned','nvoid','nvolatile','ntypedef','ncomment','ncomment_char','
    nboilerplate_comment_char','ndox_comment','ndox_comment_char','nfun_comment','
    ncpp_directive','ncpp_include','ncpp_conditional','nfun_cpp_directive','
    nfun_cpp_conditional','style_inconsistency','nfunction2','halstead_min','
    halstead_mean','halstead_median','halstead_max','halstead_sd','nfunction3','
    cyclomatic_min','cyclomatic_mean','cyclomatic_median','cyclomatic_max','
    cyclomatic_sd','nidentifier','identifier_length_min','identifier_length_mean','
    identifier_length_median','identifier_length_max','identifier_length_sd','
    unique_nidentifier','unique_identifier_length_min','unique_identifier_length_mean
    ','unique_identifier_length_median','unique_identifier_length_max','
    unique_identifier_length_sd','indentation_spacing_count','indentation_spacing_min
    ','indentation_spacing_mean','indentation_spacing_median','
    indentation_spacing_max','indentation_spacing_sd','nno_space_after_binary_op','
    nno_space_after_closing_brace','nno_space_after_comma','nno_space_after_keyword',
    'nno_space_after_opening_brace','nno_space_after_semicolon','
    nno_space_before_binary_op','nno_space_before_closing_brace','
    nno_space_before_keyword','nno_space_before_opening_brace','
    nspace_after_opening_square_bracket','nspace_after_struct_op','
    nspace_after_unary_op','nspace_at_end_of_line','nspace_before_closing_bracket','
    nspace_before_closing_square_bracket','nspace_before_comma','
    nspace_before_opening_square_bracket','nspace_before_semicolon','
    nspace_before_struct_op','nspace_after_binary_op','nspace_after_closing_brace','
    nspace_after_comma','nspace_after_keyword','nspace_after_opening_brace','
    nspace_after_semicolon','nno_space_after_struct_op','nspace_before_binary_op','
    nspace_before_closing_brace','nspace_before_keyword','nspace_before_opening_brace
    ','nno_space_before_struct_op','nno_space_after_opening_square_bracket','
    nno_space_after_unary_op','nno_space_before_closing_bracket','
    nno_space_before_closing_square_bracket','nno_space_before_comma','
    nno_space_before_opening_square_bracket','nno_space_before_semicolon','entity']
```

radonInterface.py

This module provides routines for gathering data from Radon.

```
import subprocess
import pandas as pd
from StringIO import StringIO

def runAllAnalysisOnFile(fileToMeasure):
  results = []
  for analysisType in analysisTypes:
    results.append(pd.read_json(StringIO(runAnalysisOnFile(fileToMeasure,analysisType
    ))))

  combined = pd.concat(results)
  combined = combined.transpose()

  combined = combined.reset_index()
  combined = combined.rename(columns = {'index':'entity'})

  return combined

def runAnalysisOnFile(fileToMeasure,analysisType):
  command = "radon %s %s -j" % (analysisType, fileToMeasure)
  return subprocess.check_output(command, shell=True)

def codeMetricsTable(nameLabel, dataFrame, sourceFilesDirectory, storageConnection):
    import scmData as scm
    import storage
    import dataUtilities

    scm.gitlog.switchToRevision(sourceFilesDirectory,nameLabel)

    metricsData = []
    for entityName in dataFrame['entity']:
        nextSample = runAllAnalysisOnFile(sourceFilesDirectory+entityName)
        if nextSample.shape[1] is 10:
            metricsData.append(nextSample)
```

```
    if metricsData:
        metricsData = pd.concat(metricsData)
        metricsData = dataUtilities.formatEntityNames(metricsData,
    sourceFilesDirectory)
        metricsData.drop('rank', axis=1, inplace=True)

    return metricsData

analysisTypes = ['mi', 'raw']
```

indentationInterface.py

This module provides routines for gathering indentation data.

```
from __future__ import division # To make floating division work correctly
import pandas as pd
import os
import subprocess

isWhiteSpace = -1

def indentation(line, tabsize=4):
    line = line.expandtabs(tabsize)
    # Whitespace lines get -1, since lines of code may have identation 0
    return isWhiteSpace if line.isspace() else len(line) - len(line.lstrip())

def countLeadingSpaces(inputFile, tabsize=4):
  try:
      with open(inputFile) as ifile:
          indentLengths = [indentation(line, tabsize) for line in ifile]
          return indentLengths
  except EnvironmentError:
      return [0]

def spacesToTabs(indentCounts, tabsize=4):
    return [(entry / tabsize) for entry in indentCounts]
```

```
def indentCountStats(inputFile, tabsize=4):
  spaceCounts = countLeadingSpaces(inputFile, tabsize)
  df_spaces = pd.DataFrame(spaceCounts)

  indentCounts = spacesToTabs(spaceCounts, tabsize)
  df = pd.DataFrame(indentCounts)

  mean = df[0].mean()
  std = df[0].std()
  median = df[0].median()
  max = df[0].max()

  totalLines = len(df)
  indentLines = len(df_spaces[df_spaces[0] != isWhiteSpace])

  data = [indentLines, mean, std, median, max]

  frame = pd.DataFrame(data)
  frame = frame.transpose()
  frame.columns = ['indent_lines', 'indent_mean', 'indent_sd', 'indent_median', '
    indent_max']

  return frame

def indentMetricsTable(nameLabel, dataFrame, sourceFilesDirectory, storageConnection,
     tabsize=4):
  import storage
  import scmData as scm
  import dataUtilities

  tableName = nameLabel + '_indentMetrics'
  if storage.tableExists(tableName, storageConnection):
    return storage.readTable(tableName, storageConnection)

  scm.gitlog.switchToRevision(sourceFilesDirectory,nameLabel)
  indentMetrics = pd.DataFrame()
```

```
  for entityName in dataFrame['entity']:
    nextFrame = indentCountStats(sourceFilesDirectory+entityName, tabsize)
    nextFrame['entity'] = sourceFilesDirectory+entityName
    indentMetrics = pd.concat([indentMetrics,nextFrame])

  if indentMetrics.empty:
    return None

  indentMetrics = dataUtilities.formatEntityNames(indentMetrics, sourceFilesDirectory
    )

  storage.writeTable(tableName, storageConnection, indentMetrics)
  return indentMetrics
```

dataUtilities.py

This module provides routines for the preparation of data.

```
def formatEntityNames(dataframe, sourceFilesDirectory):
  nameTrimLength = len(sourceFilesDirectory)
  dataframe['entity'] = dataframe['entity'].apply(lambda x: str(x)[nameTrimLength:])
  dataframe['entity'] = dataframe['entity'].apply(lambda x: x.replace('/','_'))
  return dataframe

def scaleFeatures(features):
  from sklearn import preprocessing as pp
  scaler = pp.StandardScaler()
  scaler.fit(features)
  scaledFeatures = scaler.transform(features)
  return scaledFeatures

def replaceWithLabelEncoded(features, featureName):
  from sklearn import preprocessing as pp
  encoder = pp.LabelEncoder()
  encoder.fit(features[featureName])
  labelledTimes = encoder.transform(features[featureName])
  features.drop([featureName], axis=1, inplace=True)
```

```
  features[featureName] = labelledTimes
  return features

def replaceWithOneHotEncoded(features, featureName):
  import pandas as pd
  oneHotNames = pd.get_dummies(features[featureName], featureName)

  features = pd.concat([features, oneHotNames], axis=1)
  features.drop(featureName, axis=1, inplace=True)

  features.columns = [name.encode('utf-8') for name in features.columns]
  return features

def addBinnedResponseCategory(data, responseVariable, binLabels):
  import pandas as pd
  binnedData, bins = pd.cut(data[responseVariable], len(binLabels), labels=False,
    retbins=True)

  from sklearn import preprocessing as pp
  lb = pp.LabelBinarizer()
  lb.fit(binnedData)

  binnedCategories = pd.DataFrame(lb.transform(binnedData))
  binnedCategories.index = data.index
  binnedCategories.columns = binLabels
  binnedDataSet = pd.concat([data, binnedCategories], axis=1)
  binnedDataSet.drop(responseVariable, axis=1, inplace=True)

  return binnedDataSet
```

storage.py

This module provides routines for the storage of gathered data.

```
import sqlite3 as sql
import pandas as pd

def openDatabase(filename):
  return sql.connect(filename)

def writeTable(tableName, connection, dataFrame):
  dataFrame.to_sql(tableName, connection, index=False)

def updateTable(tableName, dataFrame):
  dataFrame.to_sql(tableName, connection, index=False, if_exists='replace')

def tableExists(tableName, connection):
  return pd.io.sql.has_table(tableName, connection)

def readTable(tableName, connection):
  return pd.read_sql("SELECT * from " + "'" + tableName + "'", connection)

def writeFile(tableName, connection, fileData):
  connection.execute('CREATE TABLE ' + tableName + ' (thebin BLOB)' )
  connection.execute('INSERT INTO ' + tableName + ' VALUES(?)', [buffer(fileData)] )

def readFile(tableName, connection, toTemporaryFile=False):
  row = connection.execute('SELECT * FROM ' + tableName).fetchone()
  restored = str(row[0])

  if toTemporaryFile is False:
    return restored
  else:
    import tempfile
    with tempfile.NamedTemporaryFile(delete=False) as temp:
      temp.write(restored)
      return temp.name
```

dataGathering.py

This module provides routines for gathering data provided by data source interfaces.

```
import os
import scmData as scm
import storage
import pandas as pd

def processTimeData(timeData, model, responseVariable, useScaler, categoryFunction):
  print "Response variable was %s" % (responseVariable)

  if hasattr(model, 'classes_') and categoryFunction is not None:
    response = categoryFunction(timeData[responseVariable]).astype(int)
  else:
    response = timeData[responseVariable]

  features = timeData.copy(deep=True)
  features.drop(responseVariable, axis=1, inplace=True)

  import dataUtilities as utilities
  features = utilities.replaceWithLabelEncoded(features, 'time')
  features = utilities.replaceWithOneHotEncoded(features, 'entity')

  if useScaler:
    features = utilities.scaleFeatures(features)

  return dict(features = features, response = response, names = features.columns)

def gatherTimeMetrics(studyDirectory, repositoryURL, sourceFilesDirectory,
    filesToInspect, language, branch='master', skipEvery=200, replaceMissing=True):
  import pandas as pd

  if not scm.gitlog.isGitRepo(sourceFilesDirectory):
    scm.gitlog.cloneGitRepo(studyDirectory,repositoryURL)

  databasePath = studyDirectory+'databases/'
  if not os.path.exists(databasePath):
```

```
        os.makedirs(databasePath)
    storageConnection = storage.openDatabase(databasePath+'metrics.db')

    dates = scm.gitlog.makeCommitDateMapping(sourceFilesDirectory,filesToInspect,
      storageConnection,branch)

    timeDataSet = []
    previousTime = dates.iloc[0]
    for index, nextTime in dates[skipEvery::skipEvery].iterrows():
      nextTable = gatherData(studyDirectory,
                  repositoryURL,
                  sourceFilesDirectory,
                  filesToInspect,
                  nextTime['sha'],
                  previousTime['sha'],
                  language,
                  storageConnection)
      previousTime = nextTime
      if not nextTable.empty:
        nextTable['time'] = nextTime['date']
        timeDataSet.append(nextTable)

    timeMetrics = pd.concat(timeDataSet)

    if not timeMetrics.empty and replaceMissing:
      for column in timeMetrics.columns:
        if 'entity' not in column and 'time' not in column:
          timeMetrics[column].fillna(timeMetrics[column].median(), inplace=True)

    timesSampled = len(timeDataSet)

    return dict(data = timeMetrics, times = timesSampled)

def gatherData(studyDirectory, repositoryURL, sourceFilesDirectory, filesToInspect,
      nextTime, previousTime, language=None, storageConnection=None):
    if storageConnection is not None:
      databasePath = studyDirectory+'databases/'
```

```
    if not os.path.exists(databasePath):
        os.makedirs(databasePath)
    storageConnection = storage.openDatabase(databasePath+'metrics.db')

scm.gitlog.makeGitLog(sourceFilesDirectory, filesToInspect, nextTime, previousTime,
    storageConnection)

import changeMetrics as change
changeMetrics = change.codemaat.changeMetricsTable(nextTime, previousTime, 'git',
    storageConnection)

codeMetrics = None
if language is not None:
    import staticMetrics as static
    codeMetrics = static.gatherStaticMetrics(language, nextTime, changeMetrics,
    sourceFilesDirectory, storageConnection)

changeMetrics['entity'] = changeMetrics['entity'].apply(lambda x: x.replace('/','_'
    ))
changeMetrics['netchurn'] = changeMetrics['added'] - changeMetrics['deleted']
changeMetrics.drop('total-revs', axis=1, inplace=True)

allData = pd.DataFrame()
if language is not None and codeMetrics is not None:
    allData = changeMetrics.merge(codeMetrics, on='entity')

return allData
```

modelAnalysis.py

This module provides routines for the analysis of fitted models.

```
# Original idea from: http://francescopochetti.com/pythonic-cross-validation-time-
    series-pandas-scikit-learn/
def timeSeriesFolds(features, response, numberOfFolds, foldNumber):
  import numpy as np
  k = int(np.floor(float(features.shape[0]) / numberOfFolds))
  i = foldNumber
  split = float(i-1)/i

  X = features[:(k*i)]
  y = response[:(k*i)]
  index = int(np.floor(X.shape[0] * split))

  Xtr = X[:index]
  Ytr = y[:index]
  Xte = X[(index + 1):]
  Yte = y[(index + 1):]

  return dict(xtr = Xtr, ytr = Ytr, xte = Xte, yte = Yte)

def analyseModel(model, featureNames, featuresTest, responseTest, scoreOnly):
  import pandas as pd
  print "Model.score: %f" % model.score(featuresTest, responseTest)

  if scoreOnly:
    return

  if hasattr(model,'classes_') and model.classes_ is not None:
    from sklearn.metrics import accuracy_score
    print "accuracy_score: %f" % accuracy_score(responseTest, model.predict(
    featuresTest))

    from sklearn.metrics import classification_report
    print classification_report(responseTest, model.predict(featuresTest))
```

```
    try:
      from sklearn.metrics import roc_auc_score
      responsesScore = model.predict(featuresTest)
      print "roc_auc_score: %f" % roc_auc_score(responseTest,responsesScore)
    except ValueError:
        print "roc_auc_score cannot be computed for this test set"

  if hasattr(model,'feature_importances_'):
    fi = pd.Series(model.feature_importances_)
    fn = pd.Series(featureNames)
    fin = pd.concat([fn, fi], axis=1)
    fin.columns=['name','importance']
    fin = fin[fin.importance != 0]
    print fin.sort_values(by='importance',ascending=False)

def timeSeriesCV(features, featureNames, response, numberOfFolds, model, scoreOnly):
  for foldNumber in range(2, numberOfFolds+1):
    cvSplitData = timeSeriesFolds(features, response, numberOfFolds, foldNumber)

    model.fit(cvSplitData['xtr'], cvSplitData['ytr'])
    analyseModel(model, featureNames, cvSplitData['xte'], cvSplitData['yte'],
    scoreOnly)

  return model

def timeSeriesModel(data, CVfolds, responseVariable, model, useScaler=False,
    scoreOnly=True, categoryFunction=lambda r: (r > r.mean())):
  if CVfolds < 2:
    return

  import dataGathering
  timeData = dataGathering.processTimeData(data, model, responseVariable, useScaler,
    categoryFunction)
  fittedModel = timeSeriesCV(timeData['features'], timeData['names'], timeData['
    response'], CVfolds, model, scoreOnly)
  return dict(fittedModel = fittedModel, data = timeData)
```

modelRefinement.py

This module provides routines for the refinement of fitted models.

```
def updateModel(studyDirectory, features, featureNames, responses, folds, fittedModel
    , modelInstance, scoreOnly=True, visualize=False):
    selectedFeatures = selectImportantFeatures(fittedModel, features, featureNames)

    import modelAnalysis as analysis
    updatedModel = analysis.timeSeriesCV(selectedFeatures['features'],
    selectedFeatures['names'], responses, folds, modelInstance, scoreOnly=scoreOnly)

    if visualize is True:
        import presentation
        presentation.visualizeModelRefinement(studyDirectory, fittedModel,
    updatedModel, featureNames, selectedFeatures)

    return dict(updated = updatedModel, selected = selectedFeatures)

def makeAndUpdateModel(studyDirectory, data, folds, responseVariable, model,
    modelRefined=None, categoryFunction=None, scoreOnly=True, visualize=False):
    import modelAnalysis as analysis
    timeModel = analysis.timeSeriesModel(data, folds, responseVariable, model,
    categoryFunction, scoreOnly)
    return updateModel(studyDirectory, timeModel['data']['features'], timeModel['data
    ']['names'],
                       timeModel['data']['response'], folds, timeModel['fittedModel'
    ],
                       modelRefined if modelRefined is not None else model,
    scoreOnly, visualize)

def selectImportantFeatures(fittedModel, features, featureNames):
    from sklearn.feature_selection import SelectFromModel
    reducedFeatures = SelectFromModel(fittedModel, prefit=True)
    featuresSelected = reducedFeatures.transform(features)

    featureNamesSelected = []
    for featureIndex in reducedFeatures.get_support(indices=True):
```

```
        featureNamesSelected.append(featureNames[featureIndex])

    return dict(features = featuresSelected, names = featureNamesSelected)
```

presentation.py

This module provides routines for the presentation of data and models.

```
import matplotlib.pyplot as plt

def initializeFigurePlotting(width=4,height=3):
    plt.rcParams["figure.figsize"] = [width, height]

def visualizeModelRefinement(studyDirectory, fittedModel, updatedModel, featureNames,
     selectedFeatures):
    import os
    import presentation

    visualizationDirectory = studyDirectory + '/models/'
    if not os.path.exists(visualizationDirectory):
        os.makedirs(visualizationDirectory)

    if not hasattr(fittedModel, 'estimators_'):
        filename = visualizationDirectory + 'modelOld.svg'
        presentation.visualizeTree(fittedModel, featureNames, filename)
        presentation.show_svg(filename)
    elif hasattr(fittedModel, 'estimators_'):
        for estTree in updatedModel.estimators_:
            filename = visualizationDirectory + 'modelOld'+str(estTree.random_state)+
    '.svg'
            presentation.visualizeTree(estTree, featureNames, filename)
            presentation.show_svg(filename)
    if not hasattr(updatedModel, 'estimators_'):
        filename = visualizationDirectory + 'model.svg'
        presentation.visualizeTree(updatedModel, selectedFeatures['names'], filename)
        presentation.show_svg(filename)
    elif hasattr(updatedModel, 'estimators_'):
```

```
        for estTree in updatedModel.estimators_:
            filename = visualizationDirectory + 'model'+str(estTree.random_state)+'.
    svg'
            presentation.visualizeTree(estTree, selectedFeatures['names'], filename)
            presentation.show_svg(filename)

def visualizeTree(model, featureNames, fileName):
    from sklearn import tree
    from StringIO import StringIO

    dot_data = StringIO()
    tree.export_graphviz(decision_tree=model, out_file=dot_data,
                         feature_names=featureNames,
                         filled=True, rounded=True,
                         proportion=True)

    import pydot
    graph = pydot.graph_from_dot_data(dot_data.getvalue())
    graph.write_svg(fileName)

def is_interactive():
    import __main__ as main
    return not hasattr(main, '__file__')

def show_svg(filename):
  if is_interactive():
      from IPython.display import SVG, display
      display(SVG(filename))

def seaBornTest(dataSet):
  import seaborn as sb
  sb.set_context("notebook", font_scale=2.0)
  g = sb.PairGrid(dataSet.ix[:,0:9], size=5)
  g.map_diag(plt.hist)
  g.map_offdiag(plt.scatter);
```

Appendix B

Case Study Notebooks

In this appendix, we present the IPython notebooks used to carry out the case study in Chapter 7. We begin with the notebook used in the study of the Git dataset, followed by a notebook which repeats the same study with the Vim, OpenSSL, httpd and nginx datasets.

Git Data Set

```
In [1]: from os.path import expanduser
        home = expanduser("~")

        # Used to access the toolkit modules in this directory
        import toolkit
```

0.0.1 First, we gather the dataset. This is a history of static code metrics (C and indentation) and change metrics for our project.

```
In [2]: # Used to indicate where the data should be gathered and stored
        rootDirectory = home+'/Desktop/dataSets/automationTest/'

        # Call gatherTimeMetrics and measure C, Indent and Change metrics
        # on .c and .h files from the git project's repository
        metricsData = toolkit.data.gatherTimeMetrics
                        (rootDirectory, 'https://github.com/git/git',
                        rootDirectory+'git/', '*/*.c */*.h *.c *.h',
                        ['indent','c'], skipEvery=50)
```

0.0.2 How many times did we sample from?

0.0.3 How many features and samples are in our dataset?

0.0.4 How many unique source files were measured?

```
In [3]: print metricsData['times']
        print metricsData['data'].shape
        print metricsData['data']['entity'].nunique()
```

```
308
(15811, 64)
621
```

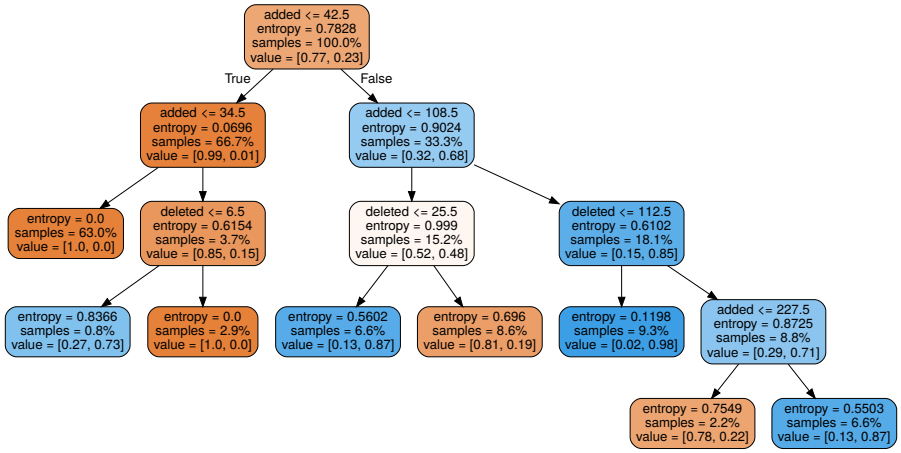
0.0.5 Let's see what affects the net churn of files

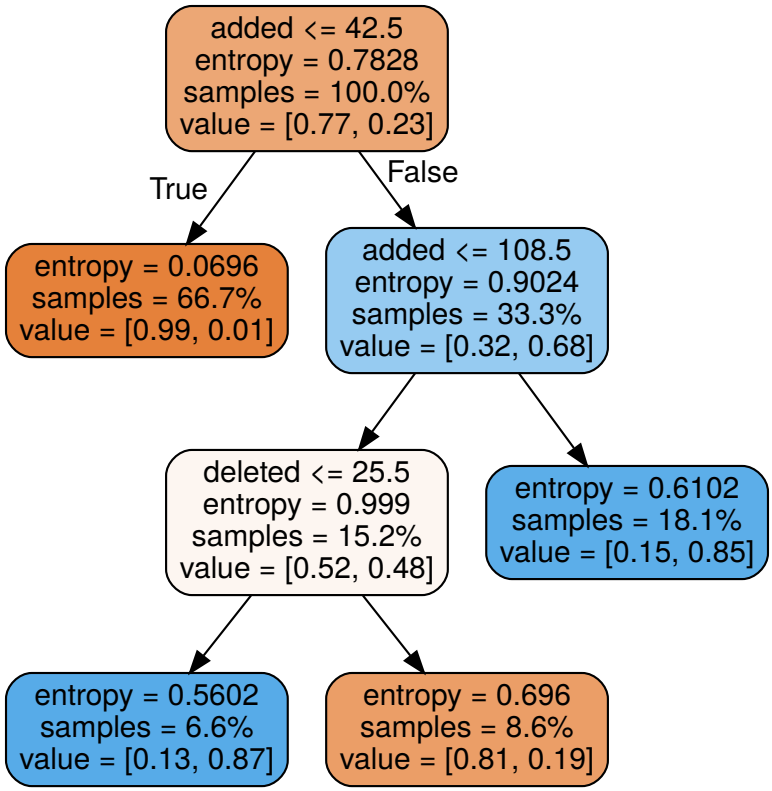
0.0.6 Which types of files have net churn above and below the mean net churn?

```
In [4]: # We instantiate the scikit-learn decision tree classification model
        # It is trained with a maximum number of leaf nodes
        # Samples are binned so as to maximize information gain at higher nodes ('entropy')

        from sklearn.tree import DecisionTreeClassifier
        modelInstance = DecisionTreeClassifier(max_leaf_nodes=8, criterion='entropy')
        modelSimpler = DecisionTreeClassifier(max_leaf_nodes=4, criterion='entropy')

        churnModel = toolkit.refinement.makeAndUpdateModel
                    (rootDirectory, metricsData['data'], 2,
                    'netchurn', modelInstance, modelSimpler,
                    visualize=True, scoreOnly=False)
```

```
Response variable was netchurn
Model.score: 0.959135
accuracy_score: 0.959135
             precision    recall  f1-score   support

          0       0.97      0.98      0.97      6110
          1       0.92      0.89      0.91      1794
```

```
avg / total       0.96      0.96      0.96      7904

roc_auc_score: 0.935571
      name  importance
3    added    0.847304
4  deleted    0.152696
Model.score: 0.949519
accuracy_score: 0.949519
             precision    recall  f1-score   support

          0       0.97      0.97      0.97      6110
          1       0.89      0.89      0.89      1794

avg / total       0.95      0.95      0.95      7904

roc_auc_score: 0.926989
      name  importance
0    added    0.896033
1  deleted    0.103967
```





0.0.7 Some observations:

- The model has very good Precision, Recall and F1-Score: net churn above/below the mean is classified very well by this model
- ROC area under curve is very high: very little compromise between false negative rate and false positive rate
- The model says that the features influencing net churn are (strongest to weakest):
  - Number of lines added
  - Number of lines deleted

- The 'net churn below the mean' class is over-represented in our data (3 times as many samples as the other class)
  - However, the model still performs well without any steps taken to address class imbalance (e.g. under/over-sampling)
- Interpretation of the visualized decision tree is straightforward:
  - 63% of samples were files with less than 34 lines added
    * These samples had net churn less than the mean

    * Some of these may be very stable files (over the history of the project)
  - The files with net churn greater than the mean had more than 108 lines added
  - Within this group, there are several subgroups with varying levels of churn

0.0.8 In the above step, we only performed 2-fold cross validation (1 training set, 1 test set)

0.0.9 How does this approach perform with more cross-validation folds in time?

In [5]:
```
# We split the data into 5 equally-sized groups,
# then perform cross-validation while gradually adding these groups to the training set

# i.e. the train-test splits are with groups of size:
# 1-4, 2-3, 3-2, 4-1

# We omit visualization of decision trees to save space,
# but they can be shown with visualize=True as above
folds = 5
churnModelMoreFolds = toolkit.refinement.makeAndUpdateModel
                        (rootDirectory, metricsData['data'],
                        folds, 'netchurn', modelInstance,
                        modelSimpler, scoreOnly=False)
```

```
Response variable was netchurn
Model.score: 0.967099
accuracy_score: 0.967099
             precision    recall  f1-score   support

          0       0.98      0.98      0.98      2607
          1       0.92      0.89      0.90       554

avg / total       0.97      0.97      0.97      3161

roc_auc_score: 0.937410
      name  importance
3    added    0.853814
4  deleted    0.146186
Model.score: 0.928820
accuracy_score: 0.928820
             precision    recall  f1-score   support

          0       0.94      0.95      0.95      2112
          1       0.90      0.88      0.89      1049

avg / total       0.93      0.93      0.93      3161

roc_auc_score: 0.917225
      name  importance
3    added    0.847395
4  deleted    0.152605
Model.score: 0.966150
accuracy_score: 0.966150
             precision    recall  f1-score   support

          0       0.97      0.98      0.98      2321
          1       0.95      0.92      0.94       840
```

```
avg / total       0.97      0.97      0.97      3161

roc_auc_score: 0.950742
      name  importance
3    added    0.855712
4  deleted    0.144288
Model.score: 0.957608
accuracy_score: 0.957608
             precision    recall  f1-score   support

          0       0.97      0.98      0.97      2634
          1       0.90      0.84      0.87       527

avg / total       0.96      0.96      0.96      3161

roc_auc_score: 0.910053
      name  importance
3    added    0.858492
4  deleted    0.141508
Model.score: 0.940525
accuracy_score: 0.940525
             precision    recall  f1-score   support

          0       0.99      0.94      0.96      2607
          1       0.77      0.95      0.85       554

avg / total       0.95      0.94      0.94      3161

roc_auc_score: 0.944043
      name  importance
0    added    0.916948
1  deleted    0.083052
Model.score: 0.915217
accuracy_score: 0.915217
             precision    recall  f1-score   support

          0       0.94      0.93      0.94      2112
          1       0.87      0.88      0.87      1049

avg / total       0.92      0.92      0.92      3161

roc_auc_score: 0.905846
      name  importance
0    added    0.894494
1  deleted    0.105506
Model.score: 0.957292
accuracy_score: 0.957292
             precision    recall  f1-score   support

          0       0.97      0.98      0.97      2321
          1       0.93      0.91      0.92       840

avg / total       0.96      0.96      0.96      3161
```

```
roc_auc_score: 0.941672
      name  importance
0    added    0.898856
1  deleted    0.101144
Model.score: 0.950332
accuracy_score: 0.950332
             precision    recall  f1-score   support

          0       0.97      0.97      0.97      2634
          1       0.85      0.86      0.85       527

avg / total       0.95      0.95      0.95      3161

roc_auc_score: 0.914036
      name  importance
0    added    0.909771
1  deleted    0.090229
```

0.0.10 Each of the subsets still exhibits class imbalance (but not with the same ratio)

0.0.11 In particular, the 2nd train-test split has the most balanced classes (2:1) among the five splits

0.0.12 Why is 'added' a much more important factor than 'deleted'?

```
In [7]: print metricsData['data']['netchurn'].mean()
        print metricsData['data']['netchurn'].var()
        print metricsData['data']['netchurn'].std()
        print metricsData['data']['netchurn'].max()
        print metricsData['data']['netchurn'].min()
```

```
34.7671874012
47557.7271358
218.077342096
4355
-3673
```

0.0.13 This codebase is *growing* in general (more added than deleted)

0.0.14 Some files must experience more churn than others. We know from some of the motivating literature that defects can be correlated with large pre-release churn.

0.0.15 Let's make some categories of binned churn data and classify them

```
In [8]: churnBinnedCategories = ['churnLow','churnMedium','churnHigh',
                                 'churnHigher', 'churnHighest']

        dataSetUpdated = toolkit.utilities.addBinnedResponseCategory
                            (metricsData['data'], 'netchurn',
                            churnBinnedCategories)

        from sklearn.tree import DecisionTreeClassifier
        modelInstance = DecisionTreeClassifier(max_leaf_nodes=8, criterion='entropy')

        churnModelCategories = toolkit.refinement.makeAndUpdateModel
                                (rootDirectory, dataSetUpdated, 2,
```

```
                                churnBinnedCategories, modelInstance,
                                modelSimpler, scoreOnly=False)
```

```
Response variable was ['churnLow', 'churnMedium', 'churnHigh', 'churnHigher', 'churnHighest']
Model.score: 0.996457
accuracy_score: 0.996457
             precision    recall  f1-score   support

          0       1.00      1.00      1.00         9
          1       0.96      0.81      0.88        97
          2       1.00      1.00      1.00      7762
          3       0.86      0.94      0.90        32
          4       0.00      0.00      0.00         4

avg / total       1.00      1.00      1.00      7904

roc_auc_score: 0.860955
           name  importance
4       deleted    0.548503
3         added    0.383638
0    age-months    0.034864
57  indent_mean    0.032995
Model.score: 0.995445
accuracy_score: 0.995445
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         9
          1       0.90      0.80      0.85        97
          2       1.00      1.00      1.00      7762
          3       0.88      0.88      0.88        32
          4       0.00      0.00      0.00         4

avg / total       0.99      1.00      0.99      7904

roc_auc_score: 0.751549
      name  importance
2  deleted      0.5991
1    added      0.4009

/home/tim/anaconda/envs/testenv/lib/python2.7/site-packages/sklearn/metrics/classification.py:1074: Und
  'precision', 'predicted', average, warn_for)
```

0.0.16 Now we're seeing something interesting. The vast majority of the files exhibit very low amounts of churn. A select few files receive most of the lines added/deleted. Does the class imbalance impact the validity of this model? Let's try more cross-validation to see.

```
In [9]: folds = 3
        churnModelCategories = toolkit.refinement.makeAndUpdateModel
                                (rootDirectory, dataSetUpdated, folds,
                                churnBinnedCategories, modelInstance,
                                modelSimpler, scoreOnly=False)
```

```
Response variable was ['churnLow', 'churnMedium', 'churnHigh', 'churnHigher', 'churnHighest']
Model.score: 0.992598
```

```
accuracy_score: 0.992598
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         3
          1       0.68      0.55      0.61        31
          2       1.00      1.00      1.00      5215
          3       0.58      0.37      0.45        19
          4       0.00      0.00      0.00         1

avg / total       0.99      0.99      0.99      5269

roc_auc_score: 0.648611
           name  importance
4       deleted    0.583342
3         added    0.279475
13    nfunction    0.063261
0    age-months    0.028592
47  halstead_sd    0.026818
2        n-revs    0.018512
Model.score: 0.994686
accuracy_score: 0.994686
             precision    recall  f1-score   support

          0       1.00      1.00      1.00         9
          1       0.96      0.81      0.88        95
          2       1.00      1.00      1.00      5129
          3       0.86      0.94      0.90        32
          4       0.00      0.00      0.00         4

avg / total       0.99      0.99      0.99      5269

roc_auc_score: 0.860285
           name  importance
4       deleted    0.547151
3         added    0.387030
0    age-months    0.035174
57  indent_mean    0.030645
Model.score: 0.990131
accuracy_score: 0.990131
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         3
          1       0.00      0.00      0.00        31
          2       1.00      1.00      1.00      5215
          3       0.76      0.68      0.72        19
          4       0.00      0.00      0.00         1

avg / total       0.99      0.99      0.99      5269

roc_auc_score: 0.636652
          name  importance
2      deleted    0.612096
1        added    0.323910
3  indent_mean    0.063994
```

```
Model.score: 0.993168
accuracy_score: 0.993168
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         9
          1       0.89      0.80      0.84        95
          2       1.00      1.00      1.00      5129
          3       0.88      0.88      0.88        32
          4       0.00      0.00      0.00         4

avg / total       0.99      0.99      0.99      5269

roc_auc_score: 0.750821
      name  importance
2  deleted    0.597757
1    added    0.402243
```

0.0.17 Let's look at this from another point of view. What characterises the files which have the most lines added?

```
In [10]: addedModel = toolkit.refinement.makeAndUpdateModel
                        (rootDirectory, metricsData['data'], 2,
                        'added', modelInstance, modelSimpler,
                        scoreOnly=False)
```

```
Response variable was added
Model.score: 0.975962
accuracy_score: 0.975962
             precision    recall  f1-score   support

          0       0.98      0.99      0.98      6270
          1       0.97      0.91      0.94      1634

avg / total       0.98      0.98      0.98      7904

roc_auc_score: 0.953173
       name  importance
6  netchurn    0.665052
3   deleted    0.317658
2    n-revs    0.017290
Model.score: 0.915992
accuracy_score: 0.915992
             precision    recall  f1-score   support

          0       0.98      0.91      0.94      6270
          1       0.73      0.95      0.82      1634

avg / total       0.93      0.92      0.92      7904

roc_auc_score: 0.927365
       name  importance
2  netchurn    0.702259
1   deleted    0.297741
```

0.0.18 Net churn and deleted lines are strongly related. What do we find if we're not allowed to use these in our decision tree?

```
In [11]: alteredData = metricsData['data'].drop(['netchurn','deleted'], axis=1)
         addedModel = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, alteredData, 2,
                         'added', modelInstance, modelSimpler,
                         scoreOnly=False)
```

```
Response variable was added
Model.score: 0.852100
accuracy_score: 0.852100
             precision    recall  f1-score   support

          0       0.88      0.94      0.91      6270
          1       0.70      0.51      0.59      1634

avg / total       0.84      0.85      0.84      7904

roc_auc_score: 0.724419
            name  importance
2         n-revs    0.921682
54  indent_lines    0.055245
12    nstatement    0.023073
Model.score: 0.852100
accuracy_score: 0.852100
             precision    recall  f1-score   support

          0       0.88      0.94      0.91      6270
          1       0.70      0.51      0.59      1634

avg / total       0.84      0.85      0.84      7904

roc_auc_score: 0.724419
     name  importance
0  n-revs           1
```

0.0.19 The model uses n-revs as the most important feature, but it does not classify '# lines added above the mean' very well

```
In [12]: alteredData2 = metricsData['data'].drop(['netchurn','deleted','n-revs'], axis=1)
         addedModel2 = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, alteredData2, 2,
                         'added', modelInstance, modelSimpler,
                         scoreOnly=False)
```

```
Response variable was added
Model.score: 0.822748
accuracy_score: 0.822748
             precision    recall  f1-score   support

          0       0.83      0.98      0.90      6270
          1       0.72      0.23      0.35      1634

avg / total       0.81      0.82      0.78      7904
```

```
roc_auc_score: 0.604104
             name  importance
1       n-authors    0.816130
2   fractal-value    0.101745
53   indent_lines    0.082125
Model.score: 0.813892
accuracy_score: 0.813892
             precision    recall  f1-score   support

          0       0.84      0.95      0.89      6270
          1       0.60      0.30      0.40      1634

avg / total       0.79      0.81      0.79      7904

roc_auc_score: 0.623862
        name  importance
0  n-authors           1
```

0.0.20 n-authors has similar problems with identifying the minority class

```
In [13]: alteredData3 = metricsData['data'].drop(['netchurn','deleted',
                                                  'n-revs','n-authors'], axis=1)

         addedModel3 = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, alteredData3, 2,
                         'added', modelInstance, modelSimpler,
                         scoreOnly=False)
```

```
Response variable was added
Model.score: 0.797950
accuracy_score: 0.797950
             precision    recall  f1-score   support

          0       0.83      0.94      0.88      6270
          1       0.52      0.24      0.33      1634

avg / total       0.76      0.80      0.77      7904

roc_auc_score: 0.593452
             name  importance
1   fractal-value    0.822226
52   indent_lines    0.082591
2             soc    0.053074
9       nfunction    0.042108
Model.score: 0.799848
accuracy_score: 0.799848
             precision    recall  f1-score   support

          0       0.80      0.99      0.89      6270
          1       0.62      0.08      0.15      1634

avg / total       0.77      0.80      0.73      7904

roc_auc_score: 0.534691
```

```
            name  importance
0  fractal-value           1
```

0.0.21 fractal-value is derived from n-revs and n-authors

0.0.22 Let's get rid of it and build a regression model for nline

0.0.23 This model will predict the sizes of files based on their other static features

```
In [14]: from sklearn.tree import DecisionTreeRegressor
         modelInstanceR = DecisionTreeRegressor(max_leaf_nodes=8)
         modelInstanceRsimpler = DecisionTreeRegressor(max_leaf_nodes=4)

         alteredData4 = metricsData['data'].drop(['netchurn','deleted','n-revs',
                                                  'n-authors','fractal-value'], axis=1)

         nlineModelR = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, alteredData4, 2,'nline',
                         modelInstanceR, modelInstanceRsimpler,
                         scoreOnly=False)
```

```
Response variable was nline
Model.score: 0.952760
                    name  importance
51          indent_lines    0.987976
50    unique_nidentifier    0.004563
285  entity_cache-tree.c    0.003730
52           indent_mean    0.001931
9             nstatement    0.001264
27                 nvoid    0.000536
Model.score: 0.963244
                   name  importance
1         indent_lines    0.991665
0   unique_nidentifier    0.004586
3  entity_cache-tree.c    0.003749
```

0.0.24 We see that the number of indented lines is a very good predictor of the number of lines

0.0.25 Is the model using indent_lines because it is correlated with nline?

```
In [15]: # We use the Spearman measure of rank correlation
         metricsData['data']['nline'].corr(metricsData['data']['indent_lines'],
                                           method='spearman')
```

0.87174000713395294

0.0.26 Let's remove indent_lines from this data set

0.0.27 Are we still able to regress on nline (and with high performance)?

```
In [16]: alteredData5 = metricsData['data'].drop(['indent_lines','nchar',
                                                  'nstatement','nidentifier'], axis=1)

         nlineModelR2 = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, alteredData5, 2, 'nline',
                         modelInstanceR, modelInstanceRsimpler,
                         scoreOnly=False)
```

```
Response variable was nline
Model.score: 0.826246
                  name  importance
40          nfunction2    0.776663
52  unique_nidentifier    0.129530
34       ncomment_char    0.074467
44        halstead_max    0.019341
Model.score: 0.780705
                 name  importance
1          nfunction2    0.852537
3  unique_nidentifier    0.147463
```

0.0.28 cqmetrics provides several measures of the 'number of functions' contained in a file (each calculated differently)

0.0.29 The model uses these to predict the size of files

```
In [17]: alteredData6 = metricsData['data'].drop(['indent_lines','nchar','nstatement','nidentifier',
                                                  'nfunction','nfunction2','nfunction3'], axis=1)

         nlineModelR3 = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, alteredData6, 2, 'nline',
                         modelInstanceR, modelInstanceRsimpler,
                         scoreOnly=False)
```

```
Response variable was nline
Model.score: 0.755902
                   name  importance
49   unique_nidentifier    0.755825
17            ninternal    0.084898
33        ncomment_char    0.052065
26              nstruct    0.035731
20                ngoto    0.026499
32             ncomment    0.023812
429   entity_log-tree.c    0.021170
Model.score: 0.720746
                 name  importance
5  unique_nidentifier    0.846590
0           ninternal    0.095093
4       ncomment_char    0.058317
```

```
In [18]: alteredData7 = metricsData['data'].drop(['indent_lines','nchar','nstatement','nidentifier',
                                                  'nfunction','nfunction2','nfunction3',
                                                  'unique_nidentifier'], axis=1)
         nlineModelR4 = toolkit.refinement.makeAndUpdateModel(rootDirectory, alteredData7, 2, 'nline',
                                                              modelInstanceR, modelInstanceRsimpler,
                                                              scoreOnly=False)
```

```
Response variable was nline
Model.score: 0.694157
                   name  importance
17            ninternal    0.713856
32             ncomment    0.101620
26              nstruct    0.091231
20                ngoto    0.041615
428  entity_log-tree.c    0.031404
```

```
43         halstead_sd    0.020273
Model.score: 0.725020
        name  importance
0  ninternal    0.802897
2    nstruct    0.102610
3   ncomment    0.094493
```

0.0.30 Our regression model of nline needs the number of (unique) identifiers, and the number of functions in a file to explain the variance in our dataset

0.0.31 Without these features, the model rapidly loses accuracy.

0.0.32 From all the features we measure, the only ones which are strong predictors of size are other measures of such (which are bound to be correlated - lines of source code necessarily add identifiers, operators, functions etc as counted by Halstead's metrics)

0.0.33 We see that the maximum Halstead complexity metric among the functions in each file is a (weak) predictor under this model. The measures from which it is calculated have a much stronger correlation with size.

0.0.34 What about modelling these measures of complexity?

```
In [19]: cycloData = metricsData['data'].drop(['cyclomatic_sd', 'cyclomatic_mean'], axis=1)

         cycloModelR = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, cycloData, 2, 'cyclomatic_max',
                         modelInstanceR, modelInstanceRsimpler,
                         scoreOnly=False)
```

```
Response variable was cyclomatic_max
Model.score: 0.639872
                     name  importance
47            halstead_max    0.806377
15              nstatement    0.080198
17  statement_nesting_mean    0.072845
22                  nconst    0.021204
58              indent_max    0.019376
Model.score: 0.579328
                     name  importance
3            halstead_max    0.865349
0              nstatement    0.088993
1  statement_nesting_mean    0.045658
```

0.0.35 Cyclomatic complexity seems to be similar to Halstead complexity for our dataset

0.0.36 The measures which are used to derive both of these are also predictors. Let's remove them and repeat..

```
In [20]: cycloData2 = metricsData['data'].drop
                     (['halstead_sd','nidentifier','halstead_mean',
                     'halstead_min','cyclomatic_sd', 'cyclomatic_mean',
                     'halstead_max','nstatement','statement_nesting_mean'],axis=1)

         cycloModelR = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, cycloData2, 2, 'cyclomatic_max',
                         modelInstanceR, modelInstanceRsimpler,
                         scoreOnly=False)
```

```
Response variable was cyclomatic_max
Model.score: 0.486573
                  name  importance
19           ninternal    0.526625
48         indent_mean    0.308199
33            ntypedef    0.064537
46  unique_nidentifier    0.038665
51          indent_max    0.032551
36      ncpp_directive    0.029424
Model.score: 0.474307
          name  importance
0    ninternal    0.630822
4  indent_mean    0.369178
```

0.0.37 Perhaps we cannot regress on cyclomatic complexity, but we can classify it in a coarse-grained categories?

```
In [21]: cycloModelC = toolkit.refinement.makeAndUpdateModel
                        (rootDirectory, cycloData2, 2, 'cyclomatic_max',
                        modelInstance, modelSimpler, scoreOnly=False)
```

```
Response variable was cyclomatic_max
Model.score: 0.845774
accuracy_score: 0.845774
             precision    recall  f1-score   support

          0       0.94      0.84      0.88      5548
          1       0.69      0.87      0.77      2356

avg / total       0.86      0.85      0.85      7904

roc_auc_score: 0.851312
                name  importance
48      indent_mean    0.614745
19        ninternal    0.248370
36   ncpp_directive    0.056529
42  halstead_median    0.040676
21            nenum    0.039680
Model.score: 0.841979
accuracy_score: 0.841979
             precision    recall  f1-score   support

          0       0.94      0.83      0.88      5548
          1       0.68      0.88      0.77      2356

avg / total       0.86      0.84      0.85      7904

roc_auc_score: 0.853126
          name  importance
4  indent_mean    0.749675
0    ninternal    0.250325
```

0.0.38 **The number of static variables and the mean indentation level of files are strong predictors of cyclomatic complexity for our dataset**

0.0.39 **This indentation predictor is similar to the findings of Hindle.**

0.0.40 **What about the 'ninternal' (static linkage) result? The files containing functions with higher cyclomatic complexity also have more variables (which are shared between functions in the same file)?**

0.0.41 **This may be starting to give some insight into our codebase. Perhaps our code overuses file-global variables together with functions which are difficult to test.**

0.0.42 **Can we model indentation? What leads to 'wider' files?**

```
In [22]: indentData = metricsData['data'].drop(['indent_sd','indent_median',
                                                'indent_max','indent_lines'],axis=1)

         indentModelR = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, indentData, 2, 'indent_mean',
                         modelInstanceR, modelInstanceRsimpler,
                         scoreOnly=False)
```

```
Response variable was indent_mean
Model.score: 0.786891
                      name  importance
47            halstead_max    0.458227
20    statement_nesting_sd    0.291467
15              nstatement    0.091504
17  statement_nesting_mean    0.052660
54           cyclomatic_sd    0.036665
8                    nchar    0.036065
46         halstead_median    0.033412
Model.score: 0.664954
                   name  importance
5          halstead_max    0.544731
3  statement_nesting_sd    0.346490
1            nstatement    0.108778
```

0.0.43 **The files with more nesting (which drives Halstead's complexity) are more indented**

0.0.44 **This tells us that our codebase uses indentation to *indicate* nesting frequently. This is typical in C programming, of course. However, there is value in this seemingly simple result: to locate the files with high syntax-driven measures of complexity in this codebase, we can use a heuristic like the level of indentation instead.**

0.0.45 **It is important to also consider that our dataset does not have other measures of complexity which have *not* been represented by this model. Halstead and McCabe's measures are dominant in the measurement of C programs, but other measures of complexity which are not strongly connected with structural *nesting* may not be predicted by indentation. In other words, indentation does not necessarily predict *complexity* - it predicts *Halstead and McCabe complexity.***

```
In [23]: indentModelC = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, indentData, 2, 'indent_mean',
                         modelInstance, modelSimpler, scoreOnly=False)
```

```
Response variable was indent_mean
Model.score: 0.942687
```

```
accuracy_score: 0.942687
             precision    recall  f1-score   support

          0       0.90      0.95      0.92      2917
          1       0.97      0.94      0.95      4987

avg / total       0.94      0.94      0.94      7904

roc_auc_score: 0.943198
                      name  importance
47            halstead_max    0.506472
20    statement_nesting_sd    0.254331
15              nstatement    0.079130
17  statement_nesting_mean    0.049194
46         halstead_median    0.042261
54           cyclomatic_sd    0.037608
9                    nline    0.031003
Model.score: 0.908148
accuracy_score: 0.908148
             precision    recall  f1-score   support

          0       0.82      0.96      0.89      2917
          1       0.98      0.88      0.92      4987

avg / total       0.92      0.91      0.91      7904

roc_auc_score: 0.919242
                   name  importance
5          halstead_max    0.602991
3  statement_nesting_sd    0.302799
1            nstatement    0.094210
```

0.0.46 We can use the DecisionTreeClassifier to bin samples above and below the mean indentation level, more effectively than we can predict the indentation level itself via regression.

0.0.47 To what degree is this true? Let's try adding more categories as before.

```
In [24]: indentBinnedCategories = ['iLow','iMedium','iHigh','iVeryHigh']

         dataSetUpdated = toolkit.utilities.addBinnedResponseCategory
                             (indentData, 'indent_mean',
                             indentBinnedCategories)

         indentModelC = toolkit.refinement.makeAndUpdateModel
                         (rootDirectory, dataSetUpdated, 2,
                         indentBinnedCategories, modelInstance,
                         modelSimpler, scoreOnly=False)
```

```
Response variable was ['iLow', 'iMedium', 'iHigh', 'iVeryHigh']
Model.score: 0.758350
accuracy_score: 0.758350
             precision    recall  f1-score   support

          0       0.80      0.78      0.79      1849
```

```
          1       0.67      0.54      0.60      2285
          2       0.78      0.90      0.84      3622
          3       0.53      0.26      0.35       148

avg / total       0.75      0.76      0.75      7904

roc_auc_score: 0.763288
                      name  importance
54           cyclomatic_sd    0.587593
20    statement_nesting_sd    0.205392
17  statement_nesting_mean    0.139729
12         line_length_max    0.067286
Model.score: 0.712171
accuracy_score: 0.712171
             precision    recall  f1-score   support

          0       0.80      0.78      0.79      1849
          1       0.54      0.74      0.62      2285
          2       0.85      0.69      0.76      3622
          3       0.00      0.00      0.00       148

avg / total       0.73      0.71      0.71      7904

roc_auc_score: 0.723178
                   name  importance
3         cyclomatic_sd    0.819947
2  statement_nesting_sd    0.180053
```

0.0.48 **This model suffers from class imbalance: with only 8 leaves, it loses accuracy when predicting the files with the most indentation. This is a common issue with decision trees being exposed by our dataset. It is not a property of our codebase or of C source code.**

0.0.49 **In this case, we can use an *ensemble model* to make up for the loss of accuracy in the minority class. However, we do so at the expense of model interpretation.**

0.0.50 **We build a random forest: a group of decision trees are made with the first decision chosen at random. The group of trees is used to classify each sample, and a majority voting scheme decides the model output.**

0.0.51 **This has the effect of creating more trees. Some may resemble the above single tree (which accurately modelled *most* of our data). Others may be highly inaccurate, except for small subsets of the data.**

0.0.52 **The training of this model is 'embarassingly parallel': we use all the available CPU cores in parallel to create our decision trees.**

0.0.53 **Forest models can be difficult to tune, and to interpret. We omit visualization of the many trees. For automated classification tasks used in a production environment (as opposed to empirical research), forests may be valuable despite their lack of interpretability.**

In [25]:
```
# For parallel construction of forest models
import psutil
cores = psutil.cpu_count()

from sklearn.ensemble import RandomForestClassifier
```

```
# We build a forest of n_estimators trees, with no restriction on the breadth/depth of trees.
modelCF = RandomForestClassifier(n_estimators=500, criterion='entropy', n_jobs=cores)

# In the updated model, each tree makes one decision
modelCFsimpler = RandomForestClassifier(n_estimators=20, max_leaf_nodes=2,
                                        criterion='entropy', n_jobs=cores)

updatedModelCF = toolkit.refinement.makeAndUpdateModel
                    (rootDirectory, dataSetUpdated, 2, indentBinnedCategories,
                    modelCF, modelCFsimpler, scoreOnly=False)
```

```
Response variable was ['iLow', 'iMedium', 'iHigh', 'iVeryHigh']
Model.score: 0.896635
accuracy_score: 0.896635
             precision    recall  f1-score   support

          0       0.95      0.95      0.95      1849
          1       0.87      0.86      0.86      2285
          2       0.92      0.92      0.92      3622
          3       1.00      0.21      0.35       148

avg / total       0.91      0.90      0.90      7904

roc_auc_score: 0.850204
                                name    importance
17            statement_nesting_mean  5.238330e-02
20              statement_nesting_sd  4.552829e-02
54                     cyclomatic_sd  4.492862e-02
45                     halstead_mean  4.375540e-02
47                      halstead_max  4.131379e-02
51                  cyclomatic_mean  3.951128e-02
48                      halstead_sd  3.873120e-02
53                    cyclomatic_max  3.710869e-02
15                        nstatement  2.834796e-02
46                  halstead_median  2.665043e-02
8                             nchar  2.527116e-02
55                     nidentifier  2.442186e-02
13                    line_length_sd  2.403311e-02
9                             nline  2.277466e-02
10                  line_length_mean  2.254443e-02
36                         ncomment  2.168801e-02
56                unique_nidentifier  2.119281e-02
37                     ncomment_char  2.082631e-02
30                           nstruct  1.978107e-02
43                        nfunction2  1.903425e-02
21                         ninternal  1.882070e-02
14                         nfunction  1.863197e-02
44                      halstead_min  1.795135e-02
12                  line_length_max  1.762424e-02
49                        nfunction3  1.746290e-02
11                line_length_median  1.725046e-02
19             statement_nesting_max  1.680899e-02
33                             nvoid  1.678905e-02
38                    ncpp_directive  1.676166e-02
```

```
39                      ncpp_include  1.661012e-02
..                               ...           ...
189          entity_builtin_bundle.c  6.423689e-08
166             entity_builtin-tag.c  6.111000e-08
557                entity_sigchain.h  5.783138e-08
654                entity_worktree.h  5.470532e-08
149          entity_builtin-reflog.c  5.442896e-08
88            entity_builtin-blame.c  4.898102e-08
309             entity_compat_msvc.h  4.673217e-08
450                   entity_mktag.c  4.616724e-08
155        entity_builtin-rev-list.c  4.552354e-08
95         entity_builtin-checkout.c  3.810218e-08
96            entity_builtin-clean.c  3.443982e-08
150          entity_builtin-remote.c  2.948679e-08
132      entity_builtin-merge-ours.c  2.538662e-08
453       entity_mozilla-sha1_sha1.h  2.528028e-08
270        entity_builtin_tar-tree.c  2.357418e-08
109     entity_builtin-fast-export.c  2.130551e-08
107       entity_builtin-diff-tree.c  2.057400e-08
112           entity_builtin-fetch.c  1.450016e-08
135           entity_builtin-merge.c  1.300473e-08
161        entity_builtin-shortlog.c  1.221451e-08
153          entity_builtin-rerere.c  1.041322e-08
169  entity_builtin-unpack-objects.c  1.031845e-08
154           entity_builtin-reset.c  9.844632e-09
642     entity_vcs-svn_string_pool.c  8.515465e-09
623      entity_update-server-info.c  8.266994e-09
163        entity_builtin-show-ref.c  7.404471e-09
567                entity_streaming.h  5.088439e-09
507            entity_receive-pack.c  4.332955e-09
514      entity_refs_files-backend.c  2.740933e-09
282      entity_builtin_write-tree.c  2.490085e-09

[490 rows x 2 columns]
Model.score: 0.325405
accuracy_score: 0.325405
             precision    recall  f1-score   support

          0       0.00      0.00      0.00      1849
          1       0.00      0.00      0.00      2285
          2       0.85      0.71      0.78      3622
          3       0.00      0.00      0.00       148

avg / total       0.39      0.33      0.36      7904

roc_auc_score: 0.575860
                      name  importance
41             halstead_sd        0.25
8                    nchar        0.15
15              nstatement        0.15
16  statement_nesting_mean        0.10
19    statement_nesting_sd        0.05
38           halstead_mean        0.05
44         cyclomatic_mean        0.05
```

```
46          cyclomatic_max        0.05
47           cyclomatic_sd        0.05
48             nidentifier        0.05
49      unique_nidentifier        0.05
```

0.0.54 **The resulting forest together uses many features. Many of these are named entities - our trees are individually modelling the files in our codebase. This showcases a possible threat to the use of forest models with our dataset: there may be a tendency to model the one-hot encoded entities.**

0.0.55 **We also note that the minority class is predicted with high Precision, but low Recall. From the high-indentation files we predict, they are predicted correctly. However, most of the high-indentation files are missed even by this model. The F1-Score (the harmonic mean of Precision and Recall) is similarly low.**

0.0.56 **Since the training of a random forest includes random choices (the initial splits in each tree), the results of this cell may vary each time it is run. The models before and after the update are likely to differ.**

Other Data Sets

```
In [7]: from os.path import expanduser
        home = expanduser("~")

        import toolkit

In [8]: def caseStudyAnalysis(dataSet):
            print dataSet['times']
            print dataSet['data'].shape
            print dataSet['data']['entity'].nunique()

            from sklearn.tree import DecisionTreeClassifier
            modelInstance = DecisionTreeClassifier(max_leaf_nodes=8, criterion='entropy')
            modelSimpler = DecisionTreeClassifier(max_leaf_nodes=4, criterion='entropy')

            churnModel = toolkit.refinement.makeAndUpdateModel
                        (rootDirectory, dataSet['data'], 2, 'netchurn',
                         modelInstance, modelSimpler, scoreOnly=False)

            churnBinnedCategories = ['churnLow','churnMedium','churnHigh',
                                     'churnHigher', 'churnHighest']

            dataSetUpdated = toolkit.utilities.addBinnedResponseCategory
                            (dataSet['data'], 'netchurn', churnBinnedCategories)

            churnModelCategories = toolkit.refinement.makeAndUpdateModel
                                   (rootDirectory, dataSetUpdated, 2, churnBinnedCategories,
                                    modelInstance, modelSimpler, scoreOnly=False)

            addedData = dataSet['data'].drop(['netchurn','deleted','n-revs','n-authors'], axis=1)

            addedModel = toolkit.refinement.makeAndUpdateModel
                            (rootDirectory, addedData, 2, 'added',
                            modelInstance, modelSimpler, scoreOnly=False)

            from sklearn.tree import DecisionTreeRegressor
            modelInstanceR = DecisionTreeRegressor(max_leaf_nodes=8)
            modelInstanceRsimpler = DecisionTreeRegressor(max_leaf_nodes=4)

            nlineData = dataSet['data'].drop(['indent_lines','nchar','nstatement',
                                              'nidentifier', 'nfunction','nfunction2',
                                              'nfunction3', 'unique_nidentifier'], axis=1)

            nlineModelR = toolkit.refinement.makeAndUpdateModel
                            (rootDirectory, nlineData, 2, 'nline',
                            modelInstanceR, modelInstanceRsimpler, scoreOnly=False)

            cycloData = dataSet['data'].drop
                        (['halstead_sd','nidentifier','halstead_mean',
                        'halstead_min', 'cyclomatic_sd', 'cyclomatic_mean',
                        'halstead_max','nstatement','statement_nesting_mean'],axis=1)

            cycloModelR = toolkit.refinement.makeAndUpdateModel
                            (rootDirectory, cycloData, 2, 'cyclomatic_max',
                            modelInstanceR, modelInstanceRsimpler, scoreOnly=False)
```

```
        indentData = dataSet['data'].drop(['indent_sd','indent_median','indent_max',
                                           'indent_lines'],axis=1)

        indentModelR = toolkit.refinement.makeAndUpdateModel
                        (rootDirectory, indentData, 2, 'indent_mean',
                        modelInstanceR, modelInstanceRsimpler, scoreOnly=False)

        indentModelC = toolkit.refinement.makeAndUpdateModel
                        (rootDirectory, indentData, 2, 'indent_mean',
                        modelInstance, modelSimpler, scoreOnly=False)

        indentBinnedCategories = ['iLow','iMedium','iHigh','iVeryHigh']

        indentDataUpdated = toolkit.utilities.addBinnedResponseCategory
                            (indentData, 'indent_mean', indentBinnedCategories)

        indentModelC = toolkit.refinement.makeAndUpdateModel
                        (rootDirectory, indentDataUpdated, 2, indentBinnedCategories,
                        modelInstance, modelSimpler, scoreOnly=False)
```

```
In [3]: rootDirectory = home+'/Desktop/dataSets/automationTest2/'

        metricsDataVim = toolkit.data.gatherTimeMetrics
                        (rootDirectory, 'https://github.com/vim/vim',
                        rootDirectory+'vim/', '*/*.c */*.h *.c *.h',
                        ['indent','c'], skipEvery=50)

        caseStudyAnalysis(metricsDataVim)
```

```
105
(2684, 64)
193
Response variable was netchurn
Model.score: 0.988069
accuracy_score: 0.988069
             precision    recall  f1-score   support

          0       0.99      0.99      0.99      1139
          1       0.97      0.95      0.96       202

avg / total       0.99      0.99      0.99      1341

roc_auc_score: 0.972614
         name  importance
3       added    0.826081
4     deleted    0.117428
8       nline    0.033219
20  ninternal    0.023272
Model.score: 0.983594
accuracy_score: 0.983594
             precision    recall  f1-score   support

          0       1.00      0.98      0.99      1139
          1       0.91      0.99      0.95       202
```

```
avg / total       0.98      0.98      0.98      1341

roc_auc_score: 0.986270
      name  importance
0    added    0.910461
1  deleted    0.089539
Response variable was ['churnLow', 'churnMedium', 'churnHigh', 'churnHigher', 'churnHighest']
Model.score: 0.976883
accuracy_score: 0.976883
             precision    recall  f1-score   support

          0       0.25      1.00      0.40         3
          1       1.00      1.00      1.00      1286
          2       0.61      0.74      0.67        34
          3       0.00      0.00      0.00        15
          4       0.00      0.00      0.00         3

avg / total       0.97      0.98      0.97      1341

roc_auc_score: 0.760412
      name  importance
4  deleted    0.704087
3    added    0.295913
Model.score: 0.976883
accuracy_score: 0.976883
             precision    recall  f1-score   support

          0       0.25      1.00      0.40         3
          1       1.00      1.00      1.00      1286
          2       0.61      0.74      0.67        34
          3       0.00      0.00      0.00        15
          4       0.00      0.00      0.00         3

avg / total       0.97      0.98      0.97      1341

roc_auc_score: 0.760412
      name  importance
1  deleted    0.704087
0    added    0.295913
Response variable was added
Model.score: 0.890380
accuracy_score: 0.890380
             precision    recall  f1-score   support

          0       0.92      0.95      0.94      1136
          1       0.66      0.57      0.61       205

avg / total       0.88      0.89      0.89      1341

roc_auc_score: 0.759398
                    name  importance
25               nstruct    0.273766
80  entity_src_channel.c    0.171765
```

```
28                 nvoid    0.167381
34          ncpp_include    0.140777
53           indent_mean    0.095277
5       line_length_mean    0.076067
54             indent_sd    0.074968
Model.score: 0.847129
accuracy_score: 0.847129
             precision    recall  f1-score   support

          0       0.85      1.00      0.92      1136
          1       0.00      0.00      0.00       205

avg / total       0.72      0.85      0.78      1341

roc_auc_score: 0.500000
                    name  importance
1               nstruct    0.446665
6  entity_src_channel.c    0.280244
2                 nvoid    0.273091
Response variable was nline
Model.score: 0.871940
                        name  importance
33             ncomment_char    0.854610
51             indent_median    0.079664
20                     ngoto    0.026522
31                  ntypedef    0.013761
145  entity_src_if_py_both.h    0.013422
50                 indent_sd    0.009573
26                   nstruct    0.002446
Model.score: 0.831832
            name  importance
2  ncomment_char    0.889481
4  indent_median    0.082914
0          ngoto    0.027605
Response variable was cyclomatic_max
Model.score: 0.755015
                      name  importance
48            indent_mean    0.545713
41             nfunction2    0.207143
18   statement_nesting_sd    0.095364
10       line_length_mean    0.055912
20                 nconst    0.040790
223      entity_src_tag.c    0.028867
9                    nline    0.026211
Model.score: 0.695804
                    name  importance
5           indent_mean    0.643363
4            nfunction2    0.244209
2  statement_nesting_sd    0.112428
Response variable was indent_mean
Model.score: 0.516921
                       name  importance
55              nidentifier    0.595959
17   statement_nesting_mean    0.258014
```

```
48              halstead_sd    0.082674
51          cyclomatic_mean    0.031725
114  entity_src_gui_beval.c    0.016481
7                  netchurn    0.015148
Model.score: 0.181251
                     name  importance
4             nidentifier    0.697867
1  statement_nesting_mean    0.302133
Response variable was indent_mean
Model.score: 0.896346
accuracy_score: 0.896346
             precision    recall  f1-score   support

          0       0.95      0.77      0.85       526
          1       0.87      0.98      0.92       815

avg / total       0.90      0.90      0.89      1341

roc_auc_score: 0.874612
                     name  importance
55            nidentifier    0.501540
17  statement_nesting_mean    0.303974
48             halstead_sd    0.091511
38          ncpp_directive    0.040305
7                 netchurn    0.032615
9                    nline    0.030055
Model.score: 0.774795
accuracy_score: 0.774795
             precision    recall  f1-score   support

          0       0.76      0.62      0.68       526
          1       0.78      0.87      0.83       815

avg / total       0.77      0.77      0.77      1341

roc_auc_score: 0.747309
                     name  importance
5             nidentifier    0.622634
2  statement_nesting_mean    0.377366
Response variable was ['iLow', 'iMedium', 'iHigh', 'iVeryHigh']
Model.score: 0.834452

/home/tim/anaconda/envs/testenv/lib/python2.7/site-packages/sklearn/metrics/classification.py:1074: Und
  'precision', 'predicted', average, warn_for)
/home/tim/anaconda/envs/testenv/lib/python2.7/site-packages/sklearn/metrics/classification.py:1076: Und
  'recall', 'true', average, warn_for)

accuracy_score: 0.834452
             precision    recall  f1-score   support

          0       0.95      0.55      0.69       409
          1       0.80      0.97      0.88       827
          2       0.87      0.90      0.88       105
          3       0.00      0.00      0.00         0
```

```
avg / total       0.85      0.83      0.82      1341

roc_auc_score cannot be computed for this test set
                      name  importance
21               ninternal    0.424810
17  statement_nesting_mean    0.189541
54           cyclomatic_sd    0.123099
13          line_length_sd    0.073001
36                ncomment    0.072022
38          ncpp_directive    0.068468
55             nidentifier    0.049059
Model.score: 0.782998
accuracy_score: 0.782998
             precision    recall  f1-score   support

          0       0.79      0.55      0.65       409
          1       0.79      0.89      0.84       827
          2       0.73      0.87      0.79       105
          3       0.00      0.00      0.00         0

avg / total       0.78      0.78      0.77      1341

roc_auc_score cannot be computed for this test set
                     name  importance
2               ninternal    0.576053
1  statement_nesting_mean    0.257022
5           cyclomatic_sd    0.166925
```

```
In [4]: rootDirectory = home+'/Desktop/dataSets/automationTest3/'

        metricsDataOpenSSL = toolkit.data.gatherTimeMetrics
                             (rootDirectory, 'https://github.com/openssl/openssl',
                             rootDirectory+'openssl/', '*/*.c */*.h *.c *.h',
                             ['indent','c'], skipEvery=50)

        caseStudyAnalysis(metricsDataOpenSSL)
```

```
180
(14042, 64)
1351
Response variable was netchurn
Model.score: 0.952707
accuracy_score: 0.952707
             precision    recall  f1-score   support

          0       0.98      0.96      0.97      5503
          1       0.87      0.92      0.89      1517

avg / total       0.95      0.95      0.95      7020

roc_auc_score: 0.939754
      name  importance
3    added    0.729123
4  deleted    0.184730
6      soc    0.086147
```

```
Model.score: 0.888889
accuracy_score: 0.888889
             precision    recall  f1-score   support

          0       0.88      0.99      0.93      5503
          1       0.94      0.52      0.67      1517

avg / total       0.89      0.89      0.88      7020

roc_auc_score: 0.755328
      name  importance
0    added    0.809231
1  deleted    0.190769
Response variable was ['churnLow', 'churnMedium', 'churnHigh', 'churnHigher', 'churnHighest']
Model.score: 0.999003
accuracy_score: 0.999003
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         0
          1       0.92      1.00      0.96        12
          2       1.00      1.00      1.00      6973
          3       0.97      0.86      0.91        35
          4       0.00      0.00      0.00         0

avg / total       1.00      1.00      1.00      7020

roc_auc_score cannot be computed for this test set
                  name  importance
3                added    0.564922
4              deleted    0.382906
0           age-months    0.027663
40  nfun_cpp_directive    0.024508
Model.score: 0.999145
accuracy_score: 0.999145
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         0
          1       0.92      1.00      0.96        12
          2       1.00      1.00      1.00      6973
          3       0.89      0.97      0.93        35
          4       0.00      0.00      0.00         0

avg / total       1.00      1.00      1.00      7020

roc_auc_score cannot be computed for this test set
      name  importance
1    added    0.593431
2  deleted    0.406569
Response variable was added
Model.score: 0.877208
accuracy_score: 0.877208
             precision    recall  f1-score   support

          0       0.92      0.94      0.93      6070
```

```
          1       0.56      0.45      0.50       950

avg / total       0.87      0.88      0.87      7020

roc_auc_score: 0.697257
            name  importance
2            soc    0.464573
52  indent_lines    0.290158
4          nline    0.202204
50   nidentifier    0.023132
57          time    0.019933
Model.score: 0.881766
accuracy_score: 0.881766
             precision    recall  f1-score   support

          0       0.92      0.95      0.93      6070
          1       0.58      0.45      0.51       950

avg / total       0.87      0.88      0.88      7020

roc_auc_score: 0.699892
           name  importance
0           soc    0.565268
3  indent_lines    0.222935
1         nline    0.211797
Response variable was nline
Model.score: 0.496206
              name  importance
32        ncomment    0.717469
18          nconst    0.081335
47  cyclomatic_max    0.080383
33   ncomment_char    0.056875
42    halstead_max    0.037526
17       ninternal    0.026413
Model.score: 0.475500
             name  importance
2        ncomment    0.839413
5  cyclomatic_max    0.094046
3   ncomment_char    0.066542
Response variable was cyclomatic_max
Model.score: 0.001411
                    name  importance
18  statement_nesting_sd    0.653397
52                  time    0.116973
22                 ngoto    0.100594
48           indent_mean    0.081902
46    unique_nidentifier    0.024196
43            nfunction3    0.022938
Model.score: -0.330379
                    name  importance
0  statement_nesting_sd    0.815025
4           indent_mean    0.102161
1                 ngoto    0.082813
Response variable was indent_mean
```

```
Model.score: 0.439061
                      name  importance
55             nidentifier    0.549958
17  statement_nesting_mean    0.155195
24                   ngoto    0.130050
20    statement_nesting_sd    0.074748
11      line_length_median    0.031618
47            halstead_max    0.029448
13          line_length_sd    0.028983
Model.score: 0.407357
                     name  importance
6             nidentifier    0.658472
2  statement_nesting_mean    0.185817
4                   ngoto    0.155711
Response variable was indent_mean
Model.score: 0.781909
accuracy_score: 0.781909
             precision    recall  f1-score   support

          0       0.75      0.92      0.82      3895
          1       0.86      0.61      0.71      3125

avg / total       0.80      0.78      0.77      7020

roc_auc_score: 0.765067
                      name  importance
55             nidentifier    0.560430
17  statement_nesting_mean    0.191027
24                   ngoto    0.115048
47            halstead_max    0.049394
20    statement_nesting_sd    0.046797
9                    nline    0.037305
Model.score: 0.807265
accuracy_score: 0.807265
             precision    recall  f1-score   support

          0       0.77      0.94      0.84      3895
          1       0.89      0.65      0.75      3125

avg / total       0.82      0.81      0.80      7020

roc_auc_score: 0.791396
                     name  importance
5             nidentifier    0.610091
1  statement_nesting_mean    0.243350
3                   ngoto    0.146560
Response variable was ['iLow', 'iMedium', 'iHigh', 'iVeryHigh']
Model.score: 0.652991
accuracy_score: 0.652991
             precision    recall  f1-score   support

          0       0.82      0.51      0.63      2653
          1       0.58      0.87      0.70      3219
          2       0.75      0.40      0.52      1102
```

```
          3       0.00      0.00      0.00        46

avg / total       0.70      0.65      0.64      7020

roc_auc_score: 0.644799
                      name  importance
55             nidentifier    0.522582
53          cyclomatic_max    0.224693
24                   ngoto    0.138377
17  statement_nesting_mean    0.062608
22                  nconst    0.051740
Model.score: 0.610256
accuracy_score: 0.610256
             precision    recall  f1-score   support

          0       0.82      0.51      0.63      2653
          1       0.55      0.89      0.68      3219
          2       0.49      0.04      0.08      1102
          3       0.00      0.00      0.00        46

avg / total       0.64      0.61      0.56      7020

roc_auc_score: 0.592987
             name  importance
4     nidentifier    0.510368
3  cyclomatic_max    0.303018
2           ngoto    0.186614
```

```
In [5]: rootDirectory = home+'/Desktop/dataSets/automationTest4/'

        metricsDataHTTPD = toolkit.data.gatherTimeMetrics
                            (rootDirectory, 'https://github.com/apache/httpd',
                            rootDirectory+'httpd/', '*/*.c */*.h *.c *.h',
                            ['indent','c'], branch='trunk', skipEvery=50)

        caseStudyAnalysis(metricsDataHTTPD)
```

```
253
(10054, 64)
1866
Response variable was netchurn
Model.score: 0.963589
accuracy_score: 0.963589
             precision    recall  f1-score   support

          0       0.98      0.97      0.98      4383
          1       0.84      0.89      0.86       643

avg / total       0.96      0.96      0.96      5026

roc_auc_score: 0.932677
       name  importance
3     added    0.493211
6       soc    0.302448
4   deleted    0.157343
```

```
61     time    0.046998
Model.score: 0.923199
accuracy_score: 0.923199
             precision    recall  f1-score   support

          0       0.92      1.00      0.96      4383
          1       0.94      0.43      0.59       643

avg / total       0.92      0.92      0.91      5026

roc_auc_score: 0.711124
      name  importance
0    added    0.494887
2      soc    0.370823
1  deleted    0.134291
Response variable was ['churnLow', 'churnMedium', 'churnHigh', 'churnHigher', 'churnHighest']
Model.score: 0.998607
accuracy_score: 0.998607
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         0
          1       0.00      0.00      0.00         1
          2       1.00      1.00      1.00      5019
          3       0.00      0.00      0.00         5
          4       0.00      0.00      0.00         1

avg / total       1.00      1.00      1.00      5026

roc_auc_score cannot be computed for this test set
                                        name  importance
4                                    deleted    0.540094
3                                      added    0.234944
8                                      nline    0.129771
12                           line_length_sd    0.055152
1702  entity_server_mpm_eventopt_eventopt.c    0.040039
Model.score: 0.998607
accuracy_score: 0.998607
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         0
          1       0.00      0.00      0.00         1
          2       1.00      1.00      1.00      5019
          3       0.00      0.00      0.00         5
          4       0.00      0.00      0.00         1

avg / total       1.00      1.00      1.00      5026

roc_auc_score cannot be computed for this test set
      name  importance
1  deleted    0.596914
0    added    0.403086
Response variable was added
Model.score: 0.911659
accuracy_score: 0.911659
```

```
             precision    recall  f1-score   support

          0       0.95      0.95      0.95      4572
          1       0.51      0.55      0.53       454

avg / total       0.91      0.91      0.91      5026

roc_auc_score: 0.747100
             name  importance
52   indent_lines    0.858392
1   fractal-value    0.054257
50    nidentifier    0.050038
2             soc    0.028608
0      age-months    0.008705
Model.score: 0.943494
accuracy_score: 0.943494
             precision    recall  f1-score   support

          0       0.95      0.99      0.97      4572
          1       0.81      0.49      0.61       454

avg / total       0.94      0.94      0.94      5026

roc_auc_score: 0.740791
            name  importance
4   indent_lines    0.915421
3    nidentifier    0.053362
1  fractal-value    0.031217
Response variable was nline
Model.score: 0.562764
              name  importance
32        ncomment    0.783351
18          nconst    0.092641
49     indent_mean    0.043828
17       ninternal    0.037122
42    halstead_max    0.027007
34  ncpp_directive    0.016051
Model.score: 0.747807
          name  importance
2     ncomment    0.844719
1       nconst    0.105412
5  indent_mean    0.049869
Response variable was cyclomatic_max
Model.score: 0.428522
                                     name  importance
9                                   nline    0.671671
17                  statement_nesting_max    0.120444
18                   statement_nesting_sd    0.057666
50                          indent_median    0.048465
37                           ncpp_include    0.046137
34                               ncomment    0.031958
1747  entity_server_mpm_prefork_prefork.c    0.023659
Model.score: 0.425198
                    name  importance
```

```
0                  nline    0.790405
1  statement_nesting_max    0.141736
2   statement_nesting_sd    0.067860
Response variable was indent_mean
Model.score: 0.404867
                     name  importance
15            nstatement    0.588895
47           halstead_max    0.266340
53         cyclomatic_max    0.076895
35               ntypedef    0.027767
11    line_length_median    0.026416
20  statement_nesting_sd    0.013687
Model.score: 0.396001
             name  importance
1      nstatement    0.659025
4    halstead_max    0.254922
5  cyclomatic_max    0.086053
Response variable was indent_mean
Model.score: 0.822324
accuracy_score: 0.822324
             precision    recall  f1-score   support

          0       0.98      0.66      0.79      2530
          1       0.74      0.98      0.85      2496

avg / total       0.86      0.82      0.82      5026

roc_auc_score: 0.823405
                   name  importance
15          nstatement    0.649739
47        halstead_max    0.228352
53      cyclomatic_max    0.069654
35            ntypedef    0.027221
11  line_length_median    0.025034
Model.score: 0.822125
accuracy_score: 0.822125
             precision    recall  f1-score   support

          0       0.94      0.69      0.80      2530
          1       0.75      0.96      0.84      2496

avg / total       0.85      0.82      0.82      5026

roc_auc_score: 0.823046
             name  importance
1      nstatement    0.653101
3    halstead_max    0.265817
4  cyclomatic_max    0.081082
Response variable was ['iLow', 'iMedium', 'iHigh', 'iVeryHigh']
Model.score: 0.655989
accuracy_score: 0.655989
             precision    recall  f1-score   support

          0       0.91      0.75      0.82      2149
```

```
          1       0.69      0.54      0.60      2345
          2       0.29      0.77      0.42       505
          3       0.52      0.96      0.68        27

avg / total       0.74      0.66      0.68      5026

roc_auc_score: 0.816754
                      name  importance
15              nstatement    0.518627
47            halstead_max    0.251471
10        line_length_mean    0.085359
45           halstead_mean    0.063752
20    statement_nesting_sd    0.052101
17  statement_nesting_mean    0.028690
Model.score: 0.717668
accuracy_score: 0.717668
             precision    recall  f1-score   support

          0       0.91      0.63      0.74      2149
          1       0.67      0.86      0.75      2345
          2       0.47      0.51      0.49       505
          3       0.00      0.00      0.00        27

avg / total       0.75      0.72      0.72      5026

roc_auc_score: 0.688515
                name  importance
1        nstatement    0.576994
5      halstead_max    0.315808
0  line_length_mean    0.107198
```

```
In [6]: rootDirectory = home+'/Desktop/dataSets/automationTest5/'

        metricsDataNginx = toolkit.data.gatherTimeMetrics
                            (rootDirectory, 'https://github.com/nginx/nginx',
                            rootDirectory+'nginx/', '*/*.c */*.h *.c *.h',
                            ['indent','c'], skipEvery=50)

        caseStudyAnalysis(metricsDataNginx)
```

```
88
(3457, 64)
481
Response variable was netchurn
Model.score: 0.936306
accuracy_score: 0.936306
             precision    recall  f1-score   support

          0       0.96      0.95      0.95      1223
          1       0.89      0.89      0.89       504

avg / total       0.94      0.94      0.94      1727

roc_auc_score: 0.923534
         name  importance
```

```
3       added    0.903592
4     deleted    0.077075
0  age-months    0.019333
Model.score: 0.929357
accuracy_score: 0.929357
             precision    recall  f1-score   support

          0       0.97      0.92      0.95      1223
          1       0.84      0.94      0.89       504

avg / total       0.93      0.93      0.93      1727

roc_auc_score: 0.932626
      name  importance
1    added    0.943084
2  deleted    0.056916
Response variable was ['churnLow', 'churnMedium', 'churnHigh', 'churnHigher', 'churnHighest']
Model.score: 0.989577
accuracy_score: 0.989577
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         0
          1       0.71      0.67      0.69        15
          2       1.00      0.99      1.00      1696
          3       0.78      0.88      0.82        16
          4       0.00      0.00      0.00         0

avg / total       0.99      0.99      0.99      1727

roc_auc_score cannot be computed for this test set
               name  importance
3             added    0.653709
4           deleted    0.250897
48       nfunction3    0.072675
50  cyclomatic_mean    0.022718
Model.score: 0.986103
accuracy_score: 0.986103
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         0
          1       0.72      0.87      0.79        15
          2       1.00      0.99      0.99      1696
          3       0.52      0.88      0.65        16
          4       0.00      0.00      0.00         0

avg / total       0.99      0.99      0.99      1727

roc_auc_score cannot be computed for this test set
      name  importance
0    added    0.722645
1  deleted    0.277355
Response variable was added
Model.score: 0.702953
accuracy_score: 0.702953
```

```
             precision    recall  f1-score   support

          0       0.77      0.83      0.80      1231
          1       0.48      0.38      0.42       496

avg / total       0.69      0.70      0.69      1727

roc_auc_score: 0.606852
                            name  importance
56                    indent_max    0.406422
3                          nchar    0.326642
44                    nfunction3    0.121963
46               cyclomatic_mean    0.038840
2                            soc    0.036018
135  entity_src_core_ngx_shmtx.c    0.035149
0                     age-months    0.034966
Model.score: 0.702953
accuracy_score: 0.702953
             precision    recall  f1-score   support

          0       0.77      0.83      0.80      1231
          1       0.48      0.38      0.42       496

avg / total       0.69      0.70      0.69      1727

roc_auc_score: 0.606852
         name  importance
5  indent_max    0.475332
2       nchar    0.382025
3  nfunction3    0.142642
Response variable was nline
Model.score: 0.486441
                     name  importance
17              ninternal    0.672762
15  statement_nesting_max    0.160217
20                  ngoto    0.065319
37     nfun_cpp_directive    0.041341
45        cyclomatic_mean    0.030700
32               ncomment    0.029661
Model.score: 0.423208
                     name  importance
1              ninternal    0.769468
0  statement_nesting_max    0.183248
4     nfun_cpp_directive    0.047284
Response variable was cyclomatic_max
Model.score: 0.444193
                     name  importance
48           indent_mean    0.601107
8                  nchar    0.160588
47          indent_lines    0.063054
45     cyclomatic_median    0.057725
18  statement_nesting_sd    0.056514
42       halstead_median    0.036461
39    nfun_cpp_directive    0.024551
```

```
Model.score: 0.384425
           name  importance
6   indent_mean    0.728836
0         nchar    0.194712
5  indent_lines    0.076452
Response variable was indent_mean
Model.score: 0.648200
                                                 name  importance
15                                          nstatement    0.807803
33                                               nvoid    0.081394
24                                               ngoto    0.029578
281  entity_src_http_modules_perl_ngx_http_perl_mod...    0.027591
17                               statement_nesting_mean    0.027068
50                                       cyclomatic_min    0.026566
Model.score: 0.635539
         name  importance
0  nstatement    0.908463
3       nvoid    0.091537
Response variable was indent_mean
Model.score: 0.920093
accuracy_score: 0.920093
             precision    recall  f1-score   support

          0       0.90      0.95      0.93       915
          1       0.94      0.88      0.91       812

avg / total       0.92      0.92      0.92      1727

roc_auc_score: 0.917936
                       name  importance
15              nstatement    0.635175
52       cyclomatic_median    0.135436
20    statement_nesting_sd    0.081594
47            halstead_max    0.043386
21               ninternal    0.042149
17  statement_nesting_mean    0.035564
46         halstead_median    0.026696
Model.score: 0.883034
accuracy_score: 0.883034
             precision    recall  f1-score   support

          0       0.83      0.98      0.90       915
          1       0.97      0.78      0.86       812

avg / total       0.90      0.88      0.88      1727

roc_auc_score: 0.877002
                   name  importance
0            nstatement    0.745331
6     cyclomatic_median    0.158924
2  statement_nesting_sd    0.095745
Response variable was ['iLow', 'iMedium', 'iHigh', 'iVeryHigh']
Model.score: 0.842501
accuracy_score: 0.842501
```

```
             precision    recall  f1-score   support

          0       0.91      0.94      0.93       817
          1       0.76      0.85      0.80       566
          2       0.82      0.61      0.70       297
          3       0.87      0.55      0.68        47

avg / total       0.84      0.84      0.84      1727

roc_auc_score: 0.838030
                      name  importance
15              nstatement    0.550109
17  statement_nesting_mean    0.348854
8                    nchar    0.045749
20    statement_nesting_sd    0.027776
21               ninternal    0.027512
Model.score: 0.675738
accuracy_score: 0.675738
             precision    recall  f1-score   support

          0       0.88      0.59      0.70       817
          1       0.54      0.81      0.65       566
          2       0.70      0.78      0.74       297
          3       0.00      0.00      0.00        47

avg / total       0.71      0.68      0.67      1727

roc_auc_score: 0.711159
                      name  importance
1               nstatement    0.631869
2  statement_nesting_mean    0.368131
```

Bibliography

[1] W. F. Tichy, "Should Computer Scientists Experiment More?" *Computer*, vol. 31, no. 5, pp. 32–40, May 1998. [Online]. Available: http://dx.doi.org/10.1109/2.675631

[2] "Greg Wilson - What We Actually Know About Software Development, and Why We Believe It's True," https://vimeo.com/9270320, accessed: 2016-04-18.

[3] K. Goseva-Popstojanova, M. Hamill, and R. Perugupalli, "Large Empirical Case Study of Architecture-based Software Reliability," in *16th IEEE International Symposium on Software Reliability Engineering*, Nov 2005, pp. 10–52.

[4] A. Begel and T. Zimmermann, "Analyze This! 145 Questions for Data Scientists in Software Engineering," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 12–23. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568233

[5] "Ten Questions For Empirical Researchers," http://neverworkintheory.org/2012/08/22/ten-questions-for-empirical-researchers.html, accessed: 2016-04-18.

[6] "Empirical Answers to Fundamental Software Engineering Questions," https://bertrandmeyer.com/2013/08/29/

empirical-answers-to-fundamental-software-engineering-questions/, accessed: 2016-04-18.

[7] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, Aug 2002.

[8] V. Basili, "The Role of Experimentation in Software Engineering: Past, Current, and Future," in *Proceedings of the 18th International Conference on Software Engineering*, Mar 1996, pp. 442–449.

[9] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec 1976.

[10] N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675–689, Sep 1999.

[11] L. Hatton, "Reexamining the Fault Density-Component Size Connection," *IEEE Software*, vol. 14, no. 2, pp. 89–97, Mar 1997.

[12] A. Hindle, M. W. Godfrey, and R. C. Holt, "Reading Beside the Lines: Using Indentation to Rank Revisions by Complexity," *Science of Computer Programming*, vol. 74, no. 7, pp. 414–429, May 2009. [Online]. Available: http://www.sciencedirect.com/science/article/B6V17-4VT14CM-1/2/e0e0ddda7661dc0b291216e2025cc9e4

[13] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA: Elsevier Science Inc., 1977.

[14] F. P. Brooks, Jr., *The Mythical Man-Month (Anniversary Ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[15] M. Conway, "How Do Committees Invent?" *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[16] N. Nagappan, B. Murphy, and V. Basili, "The Influence of Organizational Structure on Software Quality," in *ACM/IEEE 30th International Conference on Software Engineering*, May 2008, pp. 521–530.

[17] Z. Lin and J. Whitehead, "Why Power Laws? An Explanation from Fine-Grained Code Changes," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 68–75.

[18] M. E. J. Newman, "Power laws, Pareto distributions and Zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005. [Online]. Available: http://dx.doi.org/10.1080/00107510500052444

[19] T. Devine, K. Goseva-Popstajanova, S. Krishnan, R. Lutz, and J. Li, "An Empirical Study of Pre-release Software Faults in an Industrial Product Line," in *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, April 2012, pp. 181–190.

[20] N. Nagappan and T. Ball, "Use of Relative Code Churn Measures to Predict System Defect Density," in *Proceedings of the 27th International Conference on*

Software Engineering, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 284–292. [Online]. Available: http://doi.acm.org/10.1145/1062455.1062514

[21] M. D'Ambros, M. Lanza, and R. Robbes, "On the Relationship Between Change Coupling and Software Defects," in *16th Working Conference on Reverse Engineering*, Oct 2009, pp. 135–144.

[22] M. D'Ambros, M. Lanza, and H. Gall, "Fractal Figures: Visualizing Development Effort for CVS Entities," in *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, Sept 2005, pp. 1–6.

[23] H. Koziolek, B. Schlich, and C. Bilich, "A Large-Scale Industrial Case Study on Architecture-Based Software Reliability Analysis," in *21st International Symposium on Software Reliability Engineering (ISSRE)*, Nov 2010, pp. 279–288.

[24] M. Hays and J. Hayes, "The Effect of Testability on Fault Proneness: A Case Study of the Apache HTTP Server," in *23rd International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Nov 2012, pp. 153–158.

[25] D. Posnett, V. Filkov, and P. Devanbu, "Ecological Inference in Empirical Software Engineering," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov 2011, pp. 362–371.

[26] "Software Carpentry," http://software-carpentry.org/, accessed: 2016-04-18.

[27] C. Boettiger, "An Introduction to Docker for Reproducible Research," *SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, Jan 2015. [Online]. Available: http://doi.acm.org/10.1145/2723872.2723882

[28] "Examining Reproducibility in Computer Science," http://cs.brown.edu/~sk/Memos/Examining-Reproducibility/, accessed: 2016-04-18.

[29] H. Do, S. Elbaum, and G. Rothermel, "Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and Its Potential Impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, Oct 2005. [Online]. Available: http://dx.doi.org/10.1007/s10664-005-3861-2

[30] "PROMISE," http://promisedata.org/, accessed: 2016-04-18.

[31] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The Misuse of the NASA Metrics Data Program Data Sets for Automated Software Defect Prediction," in *15th Annual Conference on Evaluation Assessment in Software Engineering (EASE 2011)*, April 2011, pp. 96–103.

[32] G. Robles, "Empirical Software Engineering Research on Free/Libre/Open Source Software," in *22nd IEEE International Conference on Software Maintenance*, Sept 2006, pp. 347–350.

[33] "Github," http://github.com/, accessed: 2016-04-18.

[34] G. Gousios, "The GHTorrent Dataset and Tool Suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 233–236. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487132

[35] "Github Archive," https://www.githubarchive.org/, accessed: 2016-04-18.

[36] "Visual Intro to Machine Learning," http://www.r2d3.us/visual-intro-to-machine-learning-part-1/, accessed: 2016-04-18.

[37] A. Navada, A. N. Ansari, S. Patil, and B. A. Sonkamble, "Overview of use of decision tree algorithms in machine learning," in *IEEE Control and System Graduate Research Colloquium (ICSGRC)*, June 2011, pp. 37–42.

[38] G. E. P. Box, "Science and Statistics," *Journal of the American Statistical Association*, vol. 71, no. 356, pp. 791–799, 1976. [Online]. Available: http://www.jstor.org/stable/2286841

[39] E. Giger, M. Pinzger, and H. Gall, "Can we Predict Types of Code Changes? An Empirical Analysis," in *9th IEEE Working Conference on Mining Software Repositories (MSR)*, June 2012, pp. 217–226.

[40] J. Alonso, L. Belanche, and D. Avresky, "Predicting Software Anomalies Using Machine Learning Techniques," in *10th IEEE International Symposium on Network Computing and Applications (NCA)*, Aug 2011, pp. 163–170.

[41] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, July 2008.

[42] G. Fursin, "Collective Mind: cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning," *CoRR*, vol. abs/1308.2410, 2013. [Online]. Available: http://arxiv.org/abs/1308.2410

[43] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and the

Kepler System: Research Articles," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, pp. 1039–1065, Aug 2006. [Online]. Available: http://dx.doi.org/10.1002/cpe.v18:10

[44] "Empear," http://empear.com/, accessed: 2016-04-18.

[45] "Bitergia," http://bitergia.com/, accessed: 2016-04-18.

[46] K. Beck, *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[47] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," in *Proceedings of the 9th International Conference on Software Engineering*, ser. ICSE '87. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987, pp. 328–338. [Online]. Available: http://dl.acm.org/citation.cfm?id=41765.41801

[48] B. Boehm, "Spiral Development: Experience, Principles, and Refinements," http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5053, 2000, accessed: 2016-04-18.

[49] K. Beck, R. C. Martin, D. Thomas *et al.*, "Manifesto for Agile Software Development," http://www.agilemanifesto.org/, accessed: 2016-04-18.

[50] F. Pérez and B. E. Granger, "IPython: a System for Interactive Scientific Computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: http://ipython.org

[51] "PyData," http://pydata.org/, accessed: 2016-04-18.

[52] "SciPy," http://www.scipy.org/, accessed: 2016-04-18.

[53] "DataSchool," http://www.dataschool.io/, accessed: 2016-04-18.

[54] F. Pedregosa, G. Varoquaux *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Oct 2011.

[55] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 51–56.

[56] M. Fowler, *Refactoring: Improving the Design of Existing Code.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[57] D. North, "Patterns of Effective Delivery," https://vimeo.com/24681032, accessed: 2016-04-18.

[58] G. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *The Psychological Review*, vol. 63, no. 2, pp. 81–97, 1956. [Online]. Available: http://cogprints.org/730/

[59] J. Davis and M. Goadrich, "The Relationship Between Precision-Recall and ROC Curves," in *Proceedings of the 23rd International Conference on Machine Learning.* New York, NY, USA: ACM, 2006, pp. 233–240. [Online]. Available: http://doi.acm.org.libaccess.lib.mcmaster.ca/10.1145/1143844.1143874

[60] "Pythonic Cross Validation on Time Series," http://francescopochetti.com/pythonic-cross-validation-time-series-pandas-scikit-learn/, accessed: 2016-04-18.

[61] G. van Rossum, "Python reference manual," Amsterdam, The Netherlands, Tech. Rep., 1995.

[62] S. van der Walt, S. Colbert, and G. Varoquaux, "The NumPy Array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, March 2011.

[63] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Software: Practice and Experience*, vol. 30, no. 11, pp. 1203–1233, Sept 2000.

[64] M. Waskom, J. Vanderplas *et al.*, "seaborn: v0.7.0 (january 2016)," http://dx.doi.org/10.5281/zenodo.45133, Jan 2016.

[65] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, June 2007.

[66] "SQLite3," https://docs.python.org/2/library/sqlite3.html, accessed: 2016-04-18.

[67] "SQLite," https://www.sqlite.org/whentouse.html, accessed: 2016-04-18.

[68] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, vol. 2014, no. 239, Mar 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600239.2600241

[69] "Anaconda," https://www.continuum.io/why-anaconda, accessed: 2016-04-18.

[70] "Git," https://github.com/git/git, accessed: 2016-04-18.

[71] "GitPython," https://pypi.python.org/pypi/GitPython/, accessed: 2016-04-18.

[72] "Code-Maat," https://github.com/adamtornhill/code-maat, accessed: 2016-04-18.

[73] "cqmetrics," https://github.com/dspinellis/cqmetrics, accessed: 2016-04-18.

[74] "radon," https://github.com/rubik/radon, accessed: 2016-04-18.

[75] "indent-complexity-proxy," https://github.com/adamtornhill/indent-complexity-proxy, accessed: 2016-04-18.

[76] "Toolkit Distribution Package," http://www.cas.mcmaster.ca/~johnst3/toolkit_20June2016.tar.gz, accessed: 2016-06-20.

[77] C. Spearman, "The Proof and Measurement of Association Between Two Things," *American Journal of Psychology*, vol. 15, pp. 88–103, 1904.

[78] J. H. Friedman and U. Fayyad, "On Bias, Variance, 0/1Loss, and the Curse-of-Dimensionality," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 55–77, 1997. [Online]. Available: http://dx.doi.org/10.1023/A:1009778005914

[79] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun 1994.

[80] "Valgrind: Tool Suite," http://valgrind.org/info/tools.html, accessed: 2016-04-18.

[81] "Gnu gprof," https://sourceware.org/binutils/docs/gprof/, accessed: 2016-04-18.

[82] D. L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, Dec 1972. [Online]. Available: http://doi.acm.org/10.1145/361598.361623

[83] S. Morasca and L. Briand, "Towards a Theoretical Framework for Measuring Software Attributes," in *Fourth International Software Metrics Symposium*, Nov 1997, pp. 119–126.

[84] J. F. Smart, *Jenkins - The Definitive Guide*. Sebastopol, CA, USA: O'Reilly, July 2011.

[85] A. Guazzelli, M. Zeller, W.-C. Lin, and G. Williams, "PMML: An Open Standard for Sharing Models," *The R Journal*, vol. 1, no. 1, pp. 60–65, Jun. 2009. [Online]. Available: http://journal.r-project.org/archive/2009-1/RJournal_2009-1_Guazzelli+et+al.pdf