# Efficient Automatic Change Detection in Software Maintenance and Evolutionary Processes

SEBASTIAN HÖNEL

Licentiate Thesis in Computer and Information Science
Department of Computer Science and Media Technology
Faculty of Technology, Linnaeus University
Växjö, Sweden, 2020

**Abstract**

Software maintenance is such an integral part of its evolutionary process that it consumes much of the total resources available. Some estimate the costs of maintenance to be up to 100 times the amount of developing a software. A software not maintained builds up technical debt, and not paying off that debt timely will eventually outweigh the value of the software, if no countermeasures are undertaken. A software must adapt to changes in its environment, or to new and changed requirements. It must further receive corrections for emerging faults and vulnerabilities. Constant maintenance can prepare a software for the accommodation of future changes.

While there may be plenty of rationale for future changes, the reasons behind historical changes may not be accessible longer. Understanding change in software evolution provides valuable insights into, e.g., the quality of a project, or aspects of the underlying development process. These are worth exploiting, for, e.g., fault prediction, managing the composition of the development team, or for effort estimation models. The size of software is a metric often used in such models, yet it is not well-defined. In this thesis, we seek to establish a robust, versatile and computationally cheap metric, that quantifies the size of changes made during maintenance. We operationalize this new metric and exploit it for automated and efficient commit classification.

Our results show that the *density* of a commit, that is, the ratio between its net- and gross-size, is a metric that can replace other, more expensive metrics in existing classification models. Models using this metric represent the current state of the art in automatic commit classification. The density provides a more fine-grained and detailed insight into the types of maintenance activities in a software project.

Additional properties of commits, such as their relation or intermediate sojourn-times, have not been previously exploited for improved classification of changes. We reason about the potential of these, and suggest and implement dependent mixture- and Bayesian models that exploit joint conditional densities, models that each have their own trade-offs with regard to computational cost and complexity, and prediction accuracy. Such models can outperform well-established classifiers, such as Gradient Boosting Machines.

All of our empirical evaluation comprise large datasets, software and experiments, all of which we have published alongside the results as open-access. We have reused, extended and created datasets, and released software packages for change detection and Bayesian models used for all of the studies conducted.

## Acknowledgements

I would like to take the opportunity to thank everyone who has supported me in this journey so far; first and foremost, my supervisors, Morgan Ericsson, Welf Löwe, and Anna Wingkvist. Further, the support from my family has truly been invaluable. I would also like to thank my fellow PhD students and all other academical colleagues and acquaintances from whom I have learned so much from. A special mention goes to the Swedish Research School of Management and Information Technology (MIT) for the support and providing a critically friendly atmosphere at courses and seminars.

*Sebastian Hönel,*
May 8th 2020,
Växjö, Sweden.

# Contents

# Chapter 1

# Introduction

Software maintenance and evolution refer to a software being adapted to new requirements or a software changed to accommodate new or changed functionality, or a software receiving corrections for faults (Bendifallah, 1987). Evolutionary processes of software refer to the technical processes that enable such maintenance and to the work carried out by practitioners. These processes encompass technical artifacts required during the process, such as technical documentation (Souza, Anquetil, and K. M. d. Oliveira, 2005), source code versioning systems, or bug trackers (Antoniol et al., 2005). As long as software is maintained by people, all of these artifacts are intertwined. It is of advantage to understand *why* the software was changed and of *what kind* these changes were. This knowledge is exploited in a great number of ways, such as fault prediction (Graves et al., 2000). When combining this technical knowledge with the social aspects of the process, it opens up insights into, e.g., how to compose the development team (Gorla and Lam, 2004). Large software with millions of line of code (LOC) and countless changes warrants for an efficient and automatic detection and classification of changes. According to a field study carried out by Lientz, E. Burton Swanson, and Tompkins (1978), the maintenance of software is such an integral part of its evolutionary process that it consumes much of the total resources available. Some estimate that maintenance constitutes half or more of the costs of the entire life-cycle of a software (Boehm, 1976). Other estimations suggest it consumes up to 100 times the cost of initially developing the software (Glinz and Gall, 2010). Coincidentally, maintenance phases lack a sufficiently strong understanding.

The organization of this chapter is as follows. First, an introduction to what constitutes software maintenance and its challenges is given in Section 1.1. In Section 1.2, related and relevant concepts are laid out, followed by outlining problems and objectives in Section 1.3. The scope of this thesis is defined and delineated in Section 1.4. In Section 1.5, we summarize all of our applied research methods. We then give an overview of the research included, and how it addresses research problems and objective in Section 1.6. In Section 1.7 we list and summarize all

of the scientific contributions. The chapter is concluded by presenting conclusions and future work in Section 1.8.

## 1.1 Software Maintenance by Change

Change is a pervasive and central concept in the lifespan of almost every software. While a once implemented software does not change per se, its environment — as constituted by other hard- and software artifacts embedding it — does. Not only the environment can change, but so can the requirements of the software (Eick et al., 2001). It is then required to *adapt* the software so that it can *accommodate* the new requirements in terms of new or changed features. Software may manifest unexpected behavior or exhibit vulnerabilities to previously unknown threats. Either case requires maintenance. It becomes apparent that change is fundamental in software evolution. While change is solicited for various reasons, it is not always obvious in hindsight. This is particularly true in cases where no rationale was documented or it no longer exists.

Lehman's laws of software evolution (1980) define behaviors of such processes, and from them, a number of desirable properties can be derived. The first law — *Continuing Change* — expresses the desirability of constant adaption — maintenance — of software, to uphold a satisfactory degree of usability. In other words, a software that deteriorates by not maintaining it will eventually become obsolete and lose all of its value. Thus, many attempts since then have been made to quantify the amount of required changes. One rather recent progeny of this is *technical debt* (Sterling, 2010).

Software re-engineering and maintenance constitute a large part of acquiring knowledge about a system. Large portions of the knowledge in software systems are tacit or inaccessible. While external information and documentation may be used to gather knowledge, those are not always available. It is estimated that up to 60% of maintenance work is actually spent on comprehension (Kuhn, Ducasse, and Gírba, 2007; Abran et al., 2004). We are examining the evolutionary process of software, by focusing on detecting *change* in it. Changes are often documented imperfectly or even incorrectly (Hindle et al., 2009). Automatic and evidence-based detection and classification have the potential to relieve this situation. It is thus desirable to support the comprehension process or to aid it, by, e.g., automation.

Improved understanding of software maintenance can be exploited in a multitude of ways, such as for fault prediction and localization (Purushothaman and Perry, 2005; Leszak, Perry, and Stoll, 2002; Bell, Ostrand, and Weyuker, 2011), software quality monitoring, development process pattern detection, and planning of resources and personnel for maintenance activities (Gorla and Lam, 2004).

## 1.2 Concepts in Software Maintenance

In this section, we briefly introduce relevant concepts of software maintenance and development that are used in all of our studies.

### Software Repositories

Software development is typically aided by distributed version-control systems, also called software repositories, where one or more developers add their changes to a history, which in turn represents the entirety of a software artifact and each such change is a snapshot of it. Software repositories allow tracing and comparing changes, and each of these changes is called a commit. A commit is a set of changes, also called a changeset, that make up one entry in the repository's history. This set of changes may comprise any number of files added, removed, or edited. More formally, a commit $C_{S_t,S_{t+1}}$ is the patch or transition between two consecutive states $S_t, S_{t+1}$ of a software repository.

### Size of Software

Quantifying the size of software by measuring its source code is done in various ways. Early on, *function points* were suggested as a functional size measurement. Function points are awarded as a matter of units that correspond to the business functionality of the software. There are several ISO standards that define function points. Function points correlate strong with (gross) LOC, which gave way to criticism, as counting LOC is cheaper (Albrecht and Gaffney, 1983). Other ways of quantifying size are, e.g., counting the number of statements (Lin and Gustafson, 1988) or applying differencing of the abstract syntax tree between two versions of the source code (Fluri et al., 2007). Another example is to compare semantic changes between two versions (Jackson, Ladd, et al., 1994; Hashimoto, Mori, and Izumida, 2018).

In all of the work presented here, we base our approaches of quantification on counting affected lines in or files of the source code. We keep track of how each line or file was affected, e.g., added, deleted, modified, or renamed. The size, as described, counts the files or LOC as-is, without considering whether a line actually affects the functionality of the source code. This is what we call the *Gross-size*. The *Net-size*, on the other hand, is the gross-size minus the amount of files or LOC that did not contribute to actual changes of the software's functionality. We consider empty lines, whitespace, cloned-, dead- and unreachable code, as well as single- and multi-line comments to be without such effect. If none of the lines in a file is considered for the net-size, then neither is the file. This is expressed by the **Density**, which is the ratio between net- and gross-size. If all lines in a commit contribute to the software's functionality (i.e., no or relatively few lines without effect), then the density approaches its maximum value of 1. The lowest value

for density can hence be 0. In practice, the zero-density is a frequently occurring phenomenon and emerges in more than one out of five commits.

### Software Maintenance Activities

Software maintenance was suggested to be categorized into *adaptive*, *corrective*, or *perfective* by E Burton Swanson (1976). The reasons for adaptive changes were suggested to be changes in the data- or processing-environment (which essentially comprises the changes of the environment as described by Eick et al. (2001)). Reasons for corrective maintenance include processing-, performance-, or implementation-failures, and for perfective maintenance these include remediation of processing inefficiencies, performance enhancements, and general maintenance. This division into three principal maintenance activities is still used today (Mockus and Votta, 2000; Levin and Yehudai, 2017b). Some studies, however, add new categories or subdivide existing ones (Lin and Gustafson, 1988; Hindle et al., 2009).

## 1.3 Problem Statement and Objectives

The importance of change has a strong positive correlation with the size of a software. Even more so, the size of individual changes offers valuable insights into the software development process (Hindle et al., 2009). Software maintenance can be done in a variety of ways. The work presented here relies on software metrics, such as Lines Of Code, to quantify software and changes thereof. Such metrics measure certain aspects of the software's behavior or of its underlying source code. While studies facilitating the size of software exist, the overarching result of the applicability is inconclusive. Researchers can not exploit a well-defined size-metric for their studies. We are particularly interested in improving how the size of software is measured, so that it may be exploited in, e.g., effort estimation models or fault prediction scenarios, by automatically detecting and classifying change kinds in the source code.

Since the size of software is commonly adduced for estimating effort and productivity, it is important to define how to measure the size accurately. The International Software Benchmarking Standards Group (ISBSG)[1] provides definitions for *productivity* and *effort* in the context of software development processes. Both definitions include delivered *size*, which, since it is not well-defined here, is the *gross*-size. Estimation models using ISBSG's definitions hence are significantly depending on how the metric size is actually defined. The Constructive Cost Models I and II (COCOMO) by Boehm et al. (1981, 2000) are estimation models that are used industry-wide and based on the size of past projects, as measured by gross LOC. A corresponding metric hence must satisfy certain properties, such as robustness and versatility.

[1] ISBSG: "Software size as the main input parameter to cost estimation models," `http://isbsg.org/software-size`

In this thesis, we primarily address the following problems:

1. to establish a proper definition of a metric based on lines of code,
2. to optimize existing change-classification models by incorporation of size-related properties,
3. to enable exploitation of previously unconsidered relations of commits.

To address the first problem, we develop tools for mining of size-related properties of concurrent versioning systems and use these to gather such data from more than 1,650 *libre* (free, open source) software projects. We then study a large set of potential definitions of a metric that describes the net-size of software adequately, so that it may be used in future studies as replacement for gross LOC.

The second problem is addressed by first studying the aptitude of source code density, that is, the coefficient of net-size to gross-size, for probabilistic predictive models. We undertake systematic and iterative approaches to test the suitability of source code density in such models, validating against an existing ground truth and optimize them with regard to prediction accuracy and cost of features.

The last problem is twofold. We examine the problem of commits' properties and additional relational information, and discuss potentially suitable models. We suggest and implement Bayesian models that can incorporate all of these features, as other practical solutions are scarce. We then exploit this solution to examine the problem empirically.

What follows is a definition of overarching research-objectives that we approached in this thesis. Each objective corresponds to the research article with the same index.

**O-I:** *Investigate the variances and effects of various notions of size on effort.*

Furthermore, examine the suitability of automatically gathered notions of spent time as a potential replacement for regular time sheets, and whether these notions are accurate measurements or fair approximations of actual effort, by measuring correlations between time and density.

**O-II:** *Examine aptitude of source code density for change classification.*

By using attributes of preceding changes as additional dimensions, evaluate whether these improve prediction quality. Conduct an in-depth study of how sizes relate to maintenance activities, and how this changes with regard to net-size, then provide an overview of related work that uses size-based classification and reevaluate it in the context of source code density.

**O-III:** *Suggest, implement and evaluate multivariate Bayesian mixture models that support arbitrary dependencies.*

These models need to be designed to accommodate the use-cases of Article IV, where we attempt to reformulate the problem from a dependent mixture model to a classical multivariate mixture model, that can also be subjected to models based on joint conditional densities.

**O-IV:** *Formulate the problem of change over time in source code and suggest a practical solution.*

A survey of potential analytical and practical solutions is conducted, before we propose a transposition of the problem and suggest a solution specifically exploiting the possibilities as now available by Bayesian segmentation (Article III).

## 1.4 Scope of the Thesis

Software maintenance is an excessively large field. We limit our study to the following aspects and assumptions, (1) Mining Software Repositories; (2) Libre Software; (3) Maintenance Activities; and (4) Change Kinds. In what follows, we provide details as to these limitations and assumptions in more detail.

### Mining Software Repositories

The data-mining of software repositories has become its own discipline. It aims at extracting readily available information in such repositories. Repositories that hold information about the software are, e.g., the concurrent versioning system (holding the source code), application life-cycle management applications for tracking bugs and issues, or mailing lists. Documentation of the software is another exploitable source of actionable information. This thesis is concerned with source code repositories and analyses thereof, primarily. Further, we examine and attempt to complement other kinds of sources with properties mined from the source code and the concurrent versioning system holding it. Hence, we do not consider approaches that mine data from other sources.

### Libre Software

All of our studies examine exclusively libre software, available in the shape of software repositories from platforms such as GitHub[2]. Some early evidence suggested that libre and closed/proprietary software evolve differently under Lehman's laws (1980). Subsequent studies, such as those by Herraiz et al. (2006), then

[2]GitHub. "Open Source collaborative platform," `https://github.com/`

compared the evolutionary patterns across a great number of libre projects using both – LOC and more higher level metrics (such as number of modules and files, as used by closed software) – and demonstrate that the evolutionary patterns are the same, as measured with either kind of metric. This suggests that our findings are applicable beyond libre software, but we did not study it further within this thesis.

### Maintenance Activities

The most widely accepted categorization of software maintenance activities is the separation into *adaptive*, *corrective*, and *perfective* (E Burton Swanson, 1976; Mockus and Votta, 2000), without modification or extensions. In all of our work, we follow the subdivision into these three categories. Adaptive maintenance refers to adding new features, corrective to activities that serve the purpose of correcting faults. Perfective activities comprise actions of restructuring code to accommodate future changes. Since we report in detail how maintenance activities are distributed with regard to various constraints, such as size of a commit, our work may be applicable to studies where the three categories are further subdivided (i.e., not extended). However, these extended or different categorizations of software maintenance activities are not considered in this thesis.

### Change Kinds

Commits are often impure and developers frequently commit *tangled* changes (Kirinuki et al., 2014). We further confirmed this behavior by unveiling that developers tend to bulk-commit unrelated changes. For all intents and purposes, we assume that any commit can be purely assigned to one of the three maintenance categories. In fact, during processes of manually labeling commits by rules we reverse-engineered, it was apparent that most of the large commits with tangled changes can still be assigned to mainly one category. However, this limitation also implies that none of our developed probabilistic models can achieve perfect or almost perfect classification rates.

A more or less naturally occurring instance of tangled commits are *merge* commits. Such commits have two or more parent commits and do not serve a purpose of their own, other than literally merging the changes of their parents into one branch. While merging two or more commits of the same category would likely result in a merge commit of the same category, we cannot make any straightforward assumptions about merging commits of different maintenance activities. Only a comparatively small number of commits are merge commits. Therefore, we have deliberately excluded merge commits from our considerations.

## 1.5 Research Methods

Our work conflates theoretical and applied methods. Research in software engineering is often done evidence-based, as it incorporates human behavior and not only

technical solutions. It therefore often demands for empirical evaluations (Wohlin, Höst, and Henningsson, 2003).

In the first study, we manually collect commits from more than 1,650 open source projects. Using Git Density (**Hönel**, 2020a), we mine size and time spent manually from each project's commits. We attempted to cover a wide range of project types when we selected these. As statistical method of evaluation we used various kinds of correlations between a great number of potential notions of size and time. The chosen methods proved useful for, e.g., detecting behavioral patterns of developers.

The second study included here is based on a ground truth, a dataset previously introduced by other researchers (Levin and Yehudai, 2017a). In it, we reproduce previous results by re-engineering the proposed models for state-of-the-art automated classification of commits. We attempt a systematical and exhaustive approach to evaluate existing and new models involving source code density, and assess their predictive power in the contexts of single and multiple projects. We continue by reasoning about Markov assumptions (Markov et al., 1954) of consecutive commits and perform a preliminary study that takes the density of preceding commits into account. The positive outcome of this test gives us incentive to continue the studies of automated commit classification further, using more sophisticated approaches. A number of previous studies that take commit size into account exist. We pose research questions that put our work into direct relation to those and conclude that their results would deviate significantly if they had been conducted using source code density.

The third study is an introduction into analytical and then practical solutions to Bayesian mixture models with arbitrary dependencies. Bayesian models that do not make any naïve assumptions about the dependencies of their random variables and Hidden Markov Models (HMM) with finite and discrete state-spaces are quite similar in the sense that both use segmented data to build a posterior distribution of some random variable. While HMMs use discrete states of their state-space for segmenting, Bayesian models can use any (number and type of) arbitrary variable(s). In practice, most HMM implementations are only able to handle first-order discrete univariate data. However, for commit classification, we need to be able to deal with many variables, of mixed types. In standard naïve Bayes classification, all features are assumed to be independent to each other, so that marginal probabilities and probability densities are used (Wu, Sanner, and R. F. Oliveira, 2015). In practice, building conditional frequency tables for discrete variables is straightforward, but conditional densities for continuous random variables are almost never available. We mitigate this problem by estimating an empirical density using *Kernel Density Estimation* (KDE; Parzen, 1962). We evaluate a practical approach that eliminates these shortcomings on 16 well-known real-world datasets. A side-effect of our approach is that it allows searching the neighborhood of any sample and to measure pairwise distances between samples. We demonstrate the usefulness of this in the context of validating dimensionality-reduction techniques, such as *Principal Components Analysis* (PCA; Wold, Esbensen, and Geladi, 1987) and *t-distributed*

*stochastic neighbor embedding* (t-SNE; Maaten and Hinton, 2008).

The fourth study we include applies the new and extended Bayesian classification as introduced and evaluated earlier, but now on a dataset of commits. We take the ground-truth dataset from the second study and extend it with a number of manually labeled consecutive commits. It is necessary to reverse-engineer the rules for manual classification, as those given by Levin and Yehudai (2017) are not sufficient to adequately label commits. We measure the agreement of our labeling-process by blindly labeling a large portion of the ground-truth, until a high Kappa (Cohen, 1960) of $\geq 0.98$ is achieved. We pose conjectures as of the limitations of univariate HMMs and perform an exhaustive evaluation of such models to support these. We then reason about the nature of the problem we are trying to solve and which models may fit it. We then propose a transposition of our problem so that it can be analyzed with well-known classifiers and evaluate it thoroughly using segmentation, now available by the means of the software published alongside the third study. We discuss advantages and disadvantages of models based on the transposed problem, and lay some groundwork for choosing between dependency mixture models, and joint conditional density models.

## 1.6 Overview of the Thesis

Figure 1.1 depicts a high-level overview of the thesis. We define the three primitives *Problem* (3D box), *Objective* (rounded rectangle) and *Article* (square). The problems and objectives were previously defined in Section 1.3. The articles are summarized in the next section and represent chapters two through five of this thesis. We define three kinds of relations: Thick solid arrows indicate problems directly addressed by research objectives. Dashed arrows define a direct relationship between an objective and an article, these are one-to-one relations. Problems however may be addressed in more than one article, as indicated by the dotted arrows.

The first objective, that is to investigate the variances and effects of potential definitions of net-size-based metric (O-I), is achieved by Article I (Chapter 2). The second objective (O-II), an in-depth study of source code density and its applicability for classification problems, is achieved by Article II (Chapter 3). The third objective (O-III), proposing Bayesian models to accommodate typical problems in commit classification, as achieved by Article III (Chapter 4). The fourth and last objective (O-IV), to formulate the problem of change over time in source code, is achieved by Article IV (Chapter 5). While there is a direct one-to-one relation between objectives and articles, the single problems P2 and P3 are addressed by two and three articles in each case, respectively.

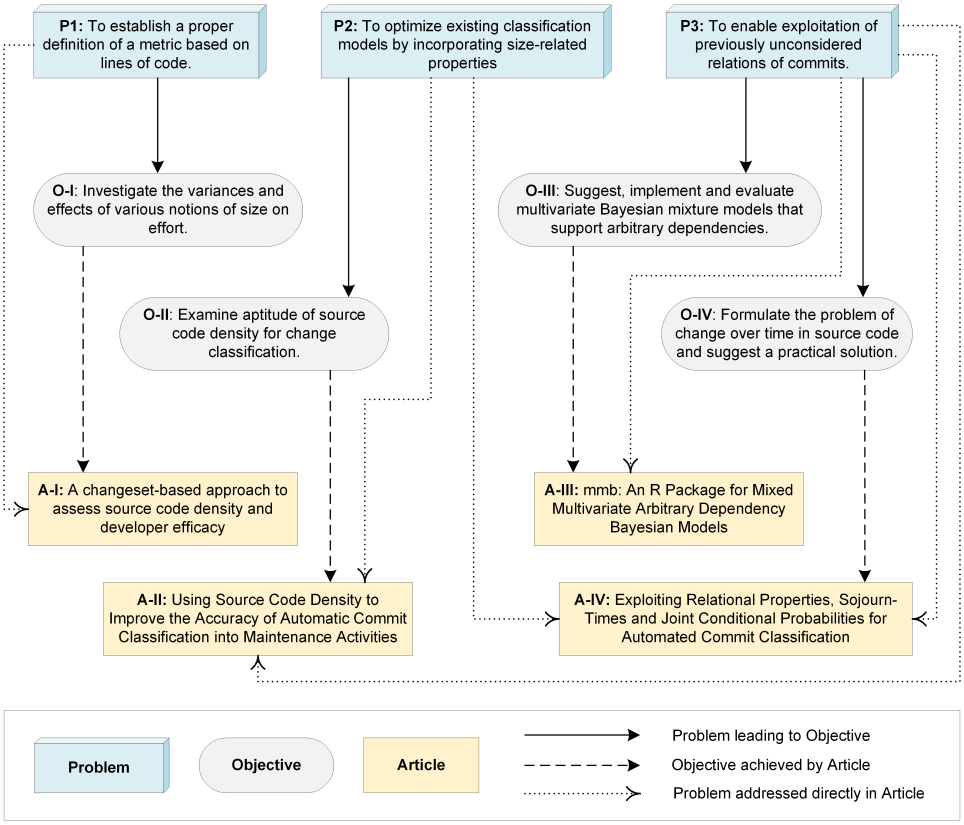


Figure 1.1: A high-level overview of Problems, Objectives and Articles in the thesis.

## 1.7 Contributions of the Thesis

In this section, we first list the scientific publications that are included in this thesis. We summarize each of the included four articles. We conclude this section with a brief list of noteworthy other contributions done during this research, but which are not directly included here.

### Scientific Publications included in this Thesis

What follows is a list of scientific publications included in this thesis. These articles constitute the main contribution. Each article is briefly summarized with regard to the research gap it addresses, the methods it uses to resolve it, and a short overview of the most significant results.

### Article I – Chapter 2

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2018). "A changeset-based approach to assess source code density and developer efficacy". In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pp. 220–221.

***Summary*** Previously, effort- and productivity-estimation models operating on the notions of size of software have been proposed. We introduce the notion of *net*-size of software, which we derive from the *gross*-size, which is determined by LOC. Software cost estimation heavily relies on accurate knowledge of how much functionality can be delivered over time. LOC however is not a well-defined metric, and we propose measuring required effort using *Source Code Density* instead, which is the ratio between net- and gross-size of code. We proceed by defining the net-size to be all lines of code that does not contain comments, white space and empty lines, dead code (such as unreachable statements), or duplicated code (type-2 clones) (Kodhai et al., 2010). The net-size is further reduced by attempting to apply various string-similarity measurements and -distances, such as n-grams or the Sørensen–Dice coefficient. For this matter, we have developed the open source tool *Git Density* (**Hönel**, 2020a). These deviations are measured between two snapshots of an entire project, divided by two, typically consecutive, commits.

Using a method for automatic detection of time spent per developer and commit[3], we gathered source code density of over 1,650 open source projects developed in Java, C#, and PHP. We evaluated correlations between notions of time spent (e.g., session length or time spent on initial commit) and density (e.g., with and without clone detection or string similarity). While none of the tested correlations were particularly strong, we found shortcomings in *Git hours* and unveiled typical behavioral patterns of developers, such as *bulk-committing* changes that would be better if they were separated. However, the deviations between the various gross- and net-sizes were significant, which was the incentive to further pursue the aptitude of source code density for purposes such as estimation of effort.

### Article II – Chapter 3

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2020c). "Using Source Code Density to Improve the Accuracy of Automatic Commit Classification into Maintenance Activities". In: *Journal of Systems and Software* (Special issue "Software Quality, Reliability, and Security"). Invited to a special issue and after a first major revision, this article is currently in stage of minor revision.

***Summary*** The reason behind changes made to a software are not always apparent. Various studies have been undertaken to automatically classify these changes.

[3]Git hours. "Estimate time spent on a Git repository," `https://github.com/kimmobrunfeldt/git-hours`, also implemented in Git Density.

Using extensive evaluations, we present how source code density can be operationalized to improve automatic commit classification. We demonstrate that (net-)size of software is an important, computationally cheap, and language-agnostic predictor. We suggest models that facilitate source code density, surpassing the current state of the art by double digits in single and cross-project scenarios.

The studies conducted in this work portray significance for a great number of use-cases, such as fault prediction, behavior detection, or the detection of flaws in the project setup or development processes. Furthermore, we observe a shift in detected activities of changes, so that earlier studies that facilitated the size of software, as measured by raw (gross) lines of code, would likely be affected by this finding. The studies were conducted using a manually collected dataset of more than 359,000 commits that was made public (**Hönel**, 2019). The work in this article represents a continuation from and build on the work done in the first article.

### Article III – Chapter 4

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2020b). "mmb: An R Package for Mixed Multivariate Arbitrary Dependency Bayesian Models". In: *Journal of Statistical Software*. Currently under review.

***Summary*** The third included article describes the challenges of Bayesian classification and regression that arise in practice, when large models with many random variables of various kinds (continuous, discrete) are required for proper prediction or regression and when there is full conditional dependency between some or all of these variables. The work done arose from the need of being able to arbitrarily segment datasets of commits and to freely model dependencies between its features. The published package for the R programming language (R Core Team, 2019) is used in the fourth article and was designed to also meet the requirements of its empirical evaluations.

We introduce a practical solution with support for mixture models that use Kernel Density Estimation (among other methods, such as the empirical cumulative distribution) to overcome the practical limits of an otherwise existing analytical solution. It supports specifying arbitrary dependencies and parallel execution. We report performance of classification and regression tasks, and compare against twelve well-established models (such as Random forests (Wright and Ziegler, 2017) and Gradient Boosting Machines (Greenwell et al., 2019)) on 16 well-known datasets (e.g., `iris`, `boston`, `sonar`, `diamonds`). We conclude that full-dependency models can outperform classical naïve approaches. This is followed by an exhaustive evaluation of the implemented hyperparameters.

We further establish and evaluate a side-effect of our implementation that can be used to search for or within a neighborhood of a sample by defining it as true statistical distance measure. This measure is then evaluated both analytically and visually, using our distance measure to validate clusters created by dimensional-

ity reduction. We demonstrate how our distance measure can outperform other measures, such as Euclidean-, Cosine-, or Tanejan-distance measures.

### Article IV – Chapter 5

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2020a). "Exploiting Relational Properties, Sojourn-Times and Joint Conditional Probabilities for Automated Commit Classification". In: *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Submitted May 8th 2020 and currently under review.

***Summary*** The last included article builds on all the work, tools, and data previously conducted and published. We demonstrate how previously not considered relations between commits and properties of those relations (e.g., dwell-times (O'Connell, Højsgaard, et al., 2011)) can be exploited to further improve automated commit classification.

The problem in its initial state, having some data that has many features and additional dimensions (continuous time) is not suitable for well-established classifiers, such as Random forest. We reason about potentially suitable models, such as Hidden (semi) Markov Models (Yu, 2010), Conditional Random Fields (Sutton, McCallum, et al., 2012), general dependent mixture models (Leroux and Puterman, 1992), and recurrent neural networks, and suggest a transformation of the problem using our previously published tool for the R programming language, to make it compatible with well-established models.

In this article, we attempt to fit 1st-, 2nd- and 3rd-order models to the problem. We implement dependent mixture models, as suggested by Visser, Speekenbrink, et al. (2010), and propose a generalization thereof. We further suggest various ways to compose models based on joint conditional densities, both stateless and stateful. Our empirical evaluation show that classical dependency models cannot deliver reliable performance, because of, e.g., *absorbing states*. The implemented models based on joint density outperform well-established classifiers, such as Random forest and Gradient Boosting Machines, by double digits for accuracy, and by almost an entire class of Kappa (+0.19).

## Other Contributions of this Thesis

In this section, we list other noteworthy contributions. These encompass software, datasets and reproducible experiments.

1. **Sebastian Hönel** (Feb. 2020a). *Git Density 2020.2: Analyze git repositories to extract the Source Code Density and other Commit Properties*. Version release-2020.2. Please check the readme.md and release notes on Github for this software. DOI: `10.5281/zenodo.2565238`. URL: `https://doi.org/10.5281/zenodo.2565238`.

2. Source Code Density experimental suite in R, available on GitHub: `https://github.com/MrShoenel/density-paper-2019-R-experiments`.

3. **Sebastian Hönel** (Mar. 2019). *359,569 commits with source code density; 1,149 commits of which have software maintenance activity labels (adaptive, corrective, perfective) [Data set].* DOI: `10.5281/zenodo.2590519`. URL: `https://doi.org/10.5281/zenodo.2590519`.

4. **Sebastian Hönel** (2020b). *mmb: Arbitrary Dependency Mixed Multivariate Bayesian Models for Inferencing, Regression, and Neighborhood Search using Joint Probabilities and Kernel Density Estimation.* R package version 1.0.0. URL: `https://cran.r-project.org/package=mmb`.

5. The datasets, empirical evaluation, reverse-engineered rules and implemented dependent mixture- and joint conditional density models are currently available from Github: `https://github.com/MrShoenel/hmm-paper-2020-R-experiments`.

## 1.8 Conclusions and Future Work

In this section, we make some preliminary concluding remarks about the state of the research done, which briefly describe the addressed problems, the proposed solutions, and the observations and lessons learned. Thereafter, we list potential directions of future work.

### Conclusions

Researchers and practitioners likewise face the problem of not having a well-defined metric that measures the size of software at their disposal. While this leads to inconsistent results or contradictory study outcomes for researchers, it results in wrong or coarse at best estimations for practitioners, when they use models that heavily rely on precise quantifications of software size (Mathew, Menzies, and Hihn, 2016). We dedicated large portions of our research to: (1) establishing source code density as a new metric to better approximate the size of software; and (2) operationalizing this metric to improve the state of the art in automated commit classification. We consider both of these to be successful and conclude with the work presented in this thesis. There is a small margin for further improvement left, but we do not consider this improvement to be significant.

Regarding the **first objective (O-I)**, we have conducted a large study of libre software and examined potential definitions of a source code density metric. We introduce the notion of net- and gross-size of software, and examines whether there are correlations between various notions of net-size and time spent. We define a computationally cheap net-size metric and conclude that further estimating net-size using string-similarities or clone-detection does not approximate the true size significantly further. Beyond that, the study unveils that automatically gathered

notions of time spent from concurrent versioning systems are not suitable for effort estimation.

The **second objective (O-II)** is wholly addressed by the second article. We conduct an in-depth study of the aptitude of source code density for automated and efficient change classification. We reproduce earlier studies and prediction models, before we enhance them. We conclude that the source code density is a significant yet cheap predictor, which can replace other features in predictive models, which are computationally more expensive to obtain. We reevaluate earlier size-based studies and point out that their results concerning how size is measured and how maintenance activities are distributed are not correct. We also exploit relational information of commits by considering the density of previous commits. This results in a significant boost towards 93% prediction accuracy of maintenance activities. At the time of publishing this article, these models represent the state of the art.

To achieve the **third objective (O-III)**, we design and develop a package for the R statistical environment (R Core Team, 2019). The suggested and implemented models were done so with regard to commit classification. However, they support arbitrary classification- and regression-problems, with mixed variables and arbitrary dependencies. We conduct an extensive empirical evaluation and conclude that the suggested models can outperform other models, such as naï Bayes or Random forest, in some cases. We further show how Bayesian segmenting can be used in cluster analysis and neighborhood search, and provide a pure metric for statistical distance that can outperform other, more traditional statistical distances, such as Euclidean or Cosine distances.

The **fourth objective (O-IV)**, to formulate the problem of change over time in source code and to suggest analytical and practical solutions, is addressed in Article IV. Previously, commit classification was done without considering state and relational properties, until we started to examine the effect of these in the second article. While size-properties of past commits are a useful predictor, neither the time between commits nor the effect of segmenting random variables based on previous commits were considered. In the fourth article, we describe the problem that adds time as an additional dimension to the problem and segmenting based on predecessors as another dimension. We suggest a transformation of the problem and a practical solution using software developed alongside Article IV. While classical dependent mixture do not perform well on this problem, models based on joint conditional densities outperform classifiers, such as Random forest, by double digits for accuracy and Kappa.

## Future Work

The desire to have accurate and automatic commit classification was inspired by the need for reducing ambiguities in software process pattern detection. Some of

these process patterns, such as *Lone Wolf*[4] or *Nine Pregnant Women*[5], may share some of the symptoms leading to them. However, without additional information, such as the kind of change, ambiguities arise which make it hard to distinguish potential patterns. Now that we have the right tools at our disposal, we plan to use these in such scenarios. We do not plan to attempt to further improve the automated commit classification as evaluated in this thesis.

Another potential future work might encompass attempts to improve automated language detection in mixed-language code-snippets. LOC is, for one reason, not a well-defined metric, as it is not language-agnostic and may thus reward more sophisticated and expressive languages less, since less code is required to express the same functionality. While in the realm of the same language this is not problematic, it might be in projects that use more than one programming- or markup-language, especially when used in the same file. Source code density is also impacted by this issue. If we had the means to isolate inhomogeneous portions of source code then we could obtain metrics for each portion separately.

Furthermore, we plan to conduct a systematic literature review, following the procedures as suggested by Kitchenham (2004). We had already started to initialize the review on the topic of "Software Incompatibility", but were forced to suspend the work. The goal of this review was to determine what the current state of the art in quantification of software incompatibilities is. Incompatibilities were to be measured between, e.g., two versions of the same software, or between two unrelated software. *Incompatibility over time*, for example, could then be a new metric to quantify *software rot* as it measures the required changes between the current software $S$ and its new, hypothetical version $S'$. Incompatibility may be measured in a rolling manner (e.g., per day/sprint/month), or between minor/major versions, or even commits. We then may be able to suggest useful applications using commit classification. However, we have been thinking about redefining the topic for the review and to start it over.

[4] "Lone Wolf" (process anti-pattern), `https://github.com/ReliSA/Software-process-antipatterns-catalogue/blob/master/catalogue/Lone-Wolf.md`

[5] "Nine Pregnant Women" (process anti-pattern), `https://github.com/ReliSA/Software-process-antipatterns-catalogue/blob/master/catalogue/Nine_Pregnant_Women.md`

# Bibliography


Abran, Alain, James W Moore, Pierre Bourque, Robert Dupuis, and L Tripp (2004). "Software engineering body of knowledge". In: *IEEE Computer Society, Angela Burgess*.

Albrecht, Allan J. and John E Gaffney (1983). "Software function, source lines of code, and development effort prediction: a software science validation". In: *IEEE transactions on software engineering* 6, pp. 639–648.

Antoniol, Giuliano, Massimiliano Di Penta, Harald Gall, and Martin Pinzger (2005). "Towards the integration of versioning systems, bug reports and source code meta-models". In: *Electronic Notes in Theoretical Computer Science* 127.3, pp. 87–99.

Bell, Robert M, Thomas J Ostrand, and Elaine J Weyuker (2011). "Does measuring code change improve fault prediction?" In: *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pp. 1–8.

Bendifallah, Salah (1987). "Understanding software maintenance work". In: *IEEE Transactions on Software Engineering* 3, pp. 311–323.

Boehm, Barry (1976). "Software Engineering". In: *IEEE Transactions on Computers* C-25.12, pp. 1226–1241.

Boehm, Barry et al. (1981). "Software engineering economics". In: *New York* 197.

Boehm, Barry, Ray Madachy, Bert Steece, et al. (2000). *Software cost estimation with Cocomo II*. Prentice Hall PTR.

Cohen, Jacob (1960). "A coefficient of agreement for nominal scales". In: *Educational and psychological measurement* 20.1, pp. 37–46.

Eick, Stephen G, Todd L Graves, Alan F Karr, J Steve Marron, and Audris Mockus (2001). "Does code decay? assessing the evidence from change management data". In: *IEEE Transactions on Software Engineering* 27.1, pp. 1–12.

Fluri, Beat, Michael Wuersch, Martin PInzger, and Harald Gall (2007). "Change distilling: Tree differencing for fine-grained source code change extraction". In: *IEEE Transactions on software engineering* 33.11, pp. 725–743.

Glinz, Martin and Harald Gall (2010). *Software Engineering (Presentation)*. Universität Zürich – Institute für Informatik. URL: `https://www.ifi.uzh.ch/dam/jcr:ffffffff-fc3b-5ce0-0000-0000051bc5f6/Kapitel_12_EvolReeng.pdf` (visited on 04/25/2020).

Gorla, Narasimhaiah and Yan Wah Lam (2004). "Who should work with whom? Building effective software project teams". In: *Communications of the ACM* 47.6, pp. 79–82.

Graves, Todd L, Alan F Karr, James S Marron, and Harvey Siy (2000). "Predicting fault incidence using software change history". In: *IEEE Transactions on software engineering* 26.7, pp. 653–661.

Greenwell, Brandon, Bradley Boehmke, Jay Cunningham, and GBM Developers (2019). *gbm: Generalized Boosted Regression Models*. R package version 2.1.5. URL: `https://CRAN.R-project.org/package=gbm`.

Hashimoto, Masatomo, Akira Mori, and Tomonori Izumida (2018). "Automated patch extraction via syntax-and semantics-aware delta debugging on source code changes". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 598–609.

Herraiz, Israel, Gregorio Robles, Jesús M González-Barahona, Andrea Capiluppi, and Juan F Ramil (2006). "Comparison between SLOCs and number of files as size metrics for software evolution analysis". In: *Conference on Software Maintenance and Reengineering (CSMR'06)*. IEEE, 8–pp.

Hindle, Abram, Daniel M German, Michael W Godfrey, and Richard C Holt (2009). "Automatic classication of large changes into maintenance categories". In: *IEEE 17th International Conference on Program Comprehension 2009 (ICPC'09)*. IEEE, pp. 30–39.

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2018). "A changeset-based approach to assess source code density and developer efficacy". In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pp. 220–221.

— (2019). "Importance and Aptitude of Source code Density for Commit Classification into Maintenance Activities". In: *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, pp. 109–120.

— (2020a). "Exploiting Relational Properties, Sojourn-Times and Joint Conditional Probabilities for Automated Commit Classification". In: *2020 35th IEEE /ACM International Conference on Automated Software Engineering (ASE)*. Submitted May 8th 2020 and currently under review.

— (2020b). "mmb: An R Package for Mixed Multivariate Arbitrary Dependency Bayesian Models". In: *Journal of Statistical Software*. Currently under review.

— (2020c). "Using Source Code Density to Improve the Accuracy of Automatic Commit Classification into Maintenance Activities". In: *Journal of Systems and Software* (Special issue "Software Quality, Reliability, and Security"). Invited to a special issue and after a first major revision, this article is currently in stage of minor revision.

**Hönel**, **Sebastian** (Mar. 2019). *359,569 commits with source code density; 1,149 commits of which have software maintenance activity labels (adaptive, corrective, perfective) [Data set]*. DOI: `10.5281/zenodo.2590519`. URL: `https://doi.org/10.5281/zenodo.2590519`.

— (Feb. 2020a). *Git Density 2020.2: Analyze git repositories to extract the Source Code Density and other Commit Properties*. Version release-2020.2. Please check the readme.md and release notes on Github for this software. DOI: 10.5281/zenodo.2565238. URL: https://doi.org/10.5281/zenodo.2565238.

— (2020b). *mmb: Arbitrary Dependency Mixed Multivariate Bayesian Models for Inferencing, Regression, and Neighborhood Search using Joint Probabilities and Kernel Density Estimation*. R package version 1.0.0. URL: https://cran.r-project.org/package=mmb.

Jackson, Daniel, David A Ladd, et al. (1994). "Semantic Diff: A Tool for Summarizing the Effects of Modifications." In: *ICSM*. Vol. 94. Citeseer, pp. 243–252.

Kirinuki, Hiroyuki, Yoshiki Higo, Keisuke Hotta, and Shinji Kusumoto (2014). "Hey! are you committing tangled changes?" In: *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, pp. 262–265.

Kitchenham, Barbara (2004). "Procedures for performing systematic reviews". In: *Keele, UK, Keele University* 33.2004, pp. 1–26.

Kodhai, E, S Kanmani, A Kamatchi, R Radhika, and B Vijaya Saranya (2010). "Detection of type-1 and type-2 code clones using textual analysis and metrics". In: *2010 International Conference on Recent Trends in Information, Telecommunication and Computing*. IEEE, pp. 241–243.

Kuhn, Adrian, Stéphane Ducasse, and Tudor Gírba (2007). "Semantic clustering: Identifying topics in source code". In: *Information and Software Technology* 49.3, pp. 230–243.

Lehman, Meir M (1980). "Programs, life cycles, and laws of software evolution". In: *Proceedings of the IEEE* 68.9, pp. 1060–1076.

Leroux, Brian G and Martin L Puterman (1992). "Maximum-penalized-likelihood estimation for independent and Markov-dependent mixture models". In: *Biometrics*, pp. 545–558.

Leszak, Marek, Dewayne E Perry, and Dieter Stoll (2002). "Classification and evaluation of defects in a project retrospective". In: *Journal of Systems and Software* 61.3, pp. 173–187.

Levin, Stanislav and Amiram Yehudai (July 2017a). *1151 commits with software maintenance activity labels (corrective, perfective, adaptive) [Data set]*. URL: http://doi.org/10.5281/zenodo.835534.

— (2017b). "Boosting automatic commit classification into maintenance activities by utilizing source code changes". In: *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 97–106.

Lientz, Bennet P, E. Burton Swanson, and Gail E Tompkins (1978). "Characteristics of application software maintenance". In: *Communications of the ACM* 21.6, pp. 466–471.

Lin, I-H and David A Gustafson (1988). "Classifying software maintenance". In: *Proceedings. Conference on Software Maintenance, 1988.* IEEE, pp. 241–247.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov, pp. 2579–2605.

Markov, Andreĭ Andreevich et al. (1954). "Theory of algorithms". In: Academy of Sciences of the USSR Moscow.

Mathew, George, Tim Menzies, and Jairus Hihn (2016). "Impacts of Bad ESP (Early Size Predictions) on Software Effort Estimation". In: *arXiv preprint arXiv:1612.03240*.

Mockus, Audris and Lawrence G Votta (2000). "Identifying Reasons for Software Changes using Historic Databases." In: *icsm*, pp. 120–130.

O'Connell, Jared, Søren Højsgaard, et al. (2011). "Hidden semi markov models for multiple observation sequences: The mhsmm package for R". In: *Journal of Statistical Software* 39.4, pp. 1–22.

Parzen, Emanuel (1962). "On estimation of a probability density function and mode". In: *The annals of mathematical statistics* 33.3, pp. 1065–1076.

Purushothaman, Ranjith and Dewayne E Perry (2005). "Toward understanding the rhetoric of small source code changes". In: *IEEE Transactions on Software Engineering* 31.6, pp. 511–526.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: `https://www.R-project.org/`.

Souza, Sergio Cozzetti B de, Nicolas Anquetil, and Káthia M de Oliveira (2005). "A study of the documentation essential to software maintenance". In: *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, pp. 68–75.

Sterling, Chris (2010). *Managing Software Debt: Building for Inevitable Change (Adobe Reader)*. Addison-Wesley Professional.

Sutton, Charles, Andrew McCallum, et al. (2012). "An introduction to conditional random fields". In: *Foundations and Trends® in Machine Learning* 4.4, pp. 267–373.

Swanson, E Burton (1976). "The dimensions of maintenance". In: *Proceedings of the 2nd International Conference on Software Engineering*. IEEE Computer Society Press, pp. 492–497.

Visser, Ingmar, Maarten Speekenbrink, et al. (2010). "depmixS4: an R package for hidden Markov models". In: *Journal of Statistical Software* 36.7, pp. 1–21.

Wohlin, Claes, Martin Höst, and Kennet Henningsson (2003). "Empirical research methods in software engineering". In: *Empirical methods and studies in software engineering*. Springer, pp. 7–23.

Wold, Svante, Kim Esbensen, and Paul Geladi (1987). "Principal component analysis". In: *Chemometrics and intelligent laboratory systems* 2.1-3, pp. 37–52.

Wright, Marvin N. and Andreas Ziegler (2017). "ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R". In: *Journal of Statistical Software* 77.1, pp. 1–17. DOI: `10.18637/jss.v077.i01`.

Wu, Ga, Scott Sanner, and Rodrigo FSC Oliveira (2015). "Bayesian model averaging naive bayes (bma-nb): Averaging over an exponential number of feature models in linear time". In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Yu, Shun-Zheng (2010). "Hidden semi-Markov models". In: *Artificial intelligence* 174.2, pp. 215–243.

# Chapter 2

# Article I: A changeset-based approach to assess source code density and developer efficacy

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2018). "A changeset-based approach to assess source code density and developer efficacy". In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pp. 220–221.

# Chapter 3

# Article II: Using Source Code Density to Improve the Accuracy of Automatic Commit Classification into Maintenance Activities

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2020c). "Using Source Code Density to Improve the Accuracy of Automatic Commit Classification into Maintenance Activities". In: *Journal of Systems and Software* (Special issue "Software Quality, Reliability, and Security"). Invited to a special issue and after a first major revision, this article is currently in stage of minor revision.

This article is an extension to the previously published conference paper:

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2019). "Importance and Aptitude of Source code Density for Commit Classification into Maintenance Activities". In: *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, pp. 109–120.

# Chapter 4

# Article III: mmb: An R Package for Mixed Multivariate Arbitrary Dependency Bayesian Models

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2020b). "mmb: An R Package for Mixed Multivariate Arbitrary Dependency Bayesian Models". In: *Journal of Statistical Software*. Currently under review.

# Chapter 5

# Article IV: Exploiting Relational Properties, Sojourn-Times and Joint Conditional Probabilities for Automated Commit Classification

**Hönel, Sebastian**, Morgan Ericsson, Welf Löwe, and Anna Wingkvist (2020a). "Exploiting Relational Properties, Sojourn-Times and Joint Conditional Probabilities for Automated Commit Classification". In: *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Submitted May 8th 2020 and currently under review.

# The Swedish Research School of Management and Information Technology

# MIT

The *Swedish Research School of Management and Information Technology* (MIT) is one of 16 national research schools supported by the Swedish Government. MIT is jointly operated by the following institutions: Blekinge Institute of Technology, Chalmers University of Technology, University of Gothenburg, Jönköping International Business School, Karlstad University, Linköping University, Linnaeus University Växjö, Lund University, Mälardalen University College, Stockholm University, Umeå University, Örebro University, and Uppsala University, host to the research school. At the Swedish Research School of Management and Information Technology (MIT), research is conducted, and doctoral education provided, in three fields: management information systems, business administration, and informatics.

## Licentiate Theses

1. **Johansson, Niklas E. (2004)** *Self-Service Recovery: Towards a Framework for Studying Service Recovery in a Self-Service Technology Context from a Management and IT Perspective,* Licentiate Thesis KUS 2004:3, Karlstad University.

2. **Ekman, Peter (2004)** *Affärssystem & Affärsrelationer: En fallstudie av en leverantörs användning av affärssystem i interaktionen med sina kunder,* Licentiate Thesis No.25, Mälardalens universitet.

3. **Wrenne, Anders (2004).** *Tjänsteplattformar: Vid utveckling av mobila tjänster inom telekommunikation,* Licentiatavhandling, KUS 2004:4, Karlstads universitet, Centrum för tjänsteforskning.

4. **Wismén, May (2004).** *Kunskapsprocesser inom hälso- och sjukvård: En studie av kunskapsintegrering mellan laboratorium och dess kunder,* Licentiatavhandling, KUS 2004:10, Karlstads universitet.

5. **Stoltz, Charlotte (2004).** *Calling for Call Centres: A Study of Call Centre Locations in a Swedish Rural Region,* Licentiate Thesis, No. 1084, IDA-EIS, Linköping University, Institute of Technology.

6. **Abelli, Björn (2004).** *Theatre Production: A System Development Process,* Licentiate thesis No. 30, Mälardalen University.

7. **Maaninen-Olsson, Eva (2004).** *Det gränslösa projektet: En studie om förmedling och skapande av kunskap i tid och rum,* Licentiatavhandling nr. 41, Företagsekonomiska institutionen, Uppsala Universitet,.

8. **Sällberg, Henrik (2004).** *On the value of customer loyalty programs: A study of point programs and switching costs,* Licentiate Thesis, No. 1116, IDA-EIS, Linköping University, Institute of Technology.

9. **Stockhult, Helén (2005).** *Medarbetaransvar - ett sätt att visa värderingar: En konceptualisering av medarbetarnas ansvar och ansvarstagande i callcenter,* Licentiatavhandling nr. 1, Örebro universitet, Institutionen för ekonomi, statistik och informatik.

10. **Vascós Palacios, Fidel (2005).** *On the information exchange between physicians and social insurance officers in the sick leave process: An Activity Theoretical perspective*, Licentiate Thesis, No. 1165, IDA-EIS, Linköping University, Institute of Technology.

11. **Keller, Christina (2005).** *Virtual Learning Environments in higher education:. A study of students' acceptance of educational technology,* Licentiate Thesis, No. 1167, IDA-EIS, Linköping University, Institute of Technology.

12. **Ahlström, Petter (2005),** *Affärsstrategier för seniorbostadsmarknaden,* Licentiatavhandling, No. 1172, IDA-EIS, Linköpings universitet, Tekniska Högskolan.

13. **Dahlin, Peter (2005).** *Structural Change of Business Networks: Developing a Structuration Technique,* Licentiate Thesis No. 49, Mälardalen University.

14. **Granebring, Annika (2005).** *ERP Migration Structure : An Innovation Process Perspective,* Licentiate Thesis No. 50, Mälardalen University.

15. **Cöster, Mathias (2005).** *Beyond IT and Productivity: How Digitization Transformed the Graphic Industry,* Licentiate Thesis, No. 1183, IDA-EIS, Linköping University, Institute of Technology.

16. **Horzella, Åsa (2005).** *Beyond IT and Productivity: Effects of Digitized Information Flows in Grocery Distribution,* Licentiate Thesis, No. 1184, IDA-EIS, Linköping University, Institute of Technology.

17. **Kollberg, Maria (2005).** *Beyond IT and Productivity: Effects of Digitized Information Flows in the Logging Industry,* Licentiate Thesis, No. 1185, IDA-EIS, Linköping University, Institute of Technology.

18. **Hansson, Magnus (2005).** *From Dusk till Dawn: Three Essays on Organizational Closedowns,* Licentiate Thesis, No. 3, Örebro University.

19. **Verma, Sanjay (2005).** *Product's Newness and Benefits to the Firm: A qualitative study from the perspective of firms developing and marketing computer software products*, Licentiate thesis, No. 54, Mälardalen University.

20. **Sundén, Susanne & Wicander, Gudrun (2005).** *Information and Communication Technology Applied for Developing Countries in a Rural Context: Towards a Framework for Analyzing Factors Influencing Sustainable Use*, Licentiate thesis, KUS 2006:69, Karlstad University.

21. **Käll, Andreas (2005).** *Översättningar av en managementmodell: En studie av införandet av Balanced Scorecard i ett landsting,* Licentiatavhandling, No.1209, IDA-EIS, Linköpings universitet, Tekniska Högskolan.

22. **Mihailescu, Daniela (2006).** *Implementation Methodology In Action: A study of an Enterprise Systems implementation methodology,* Licentiate Thesis, No.1233, IDA-EIS, Linköping University, Institute of Technology.

23. **Flodström, Raquel (2006).** *A Framework for the Strategic Management of Information Technology*, Licentiate Thesis, No.1272, IDA-EIS, Linköping University, Institute of Technology.

24. **Werelius, Sofie (2006).** *Consumer Business Relationship with Retailer and Etailer for the Purchase of Clothing: A Network Perspective*, Licentiate Thesis No. 45, Uppsala University, Department of Business Studies.

25. **Fryk, Pontus (2007).** *Beyond IT and Productivity: Effects of Digitized Information Flows in Health Care*, Licentiate Thesis, No. 1328, Linköping University, Institute of Technology.

26. **Sandström, Sara (2008).** *Technology-based service experiences: A study of the functional and emotional dimensions of telecom services*, Licentiate Thesis, KUS 2008:3, Karlstad University, Faculty of Economic Sciences, Communication and IT**.**

27. **Lundmark, Erik (2008).** *Organisational Adoption of Innovations: Management Practices and IT*, Licentiate Thesis, No. 1352, Linköping University, Institute of Technology.

28. **Anjou, Anette (2008).** *Scanias framgång: Betydelsen av strategisk kongruens och integrerad styrning*, Licentiatavhandling, No. 1364, Linköpings universitet, Tekniska högskolan.

29. **Numminen, Emil (2008).** *Software Investments under Uncertainty: Modeling Intangible Consequences as a Stochastic Process*, Licentiate Dissertation Series, No. 2008:7, Blekinge Institute of Technology.

30. **Bergqvist, Linda (2008).** *A Conceptual Framework for Studying the Successful Outcome of the IS Outsourcing Process from a Relationship Perspective*, Licentiate Thesis, KUS 2008:30, Karlstad University, Information Systems, Faculty of Economic Sciences, Communication and IT.

31. **Wingkvist, Anna (2008).** *The Quest for Equilibrium. Towards an Understanding of Scalability and Sustainability for Mobile Learning*, Licentiate Thesis, No. 08118, Växjö University, Center for Learning and Knowledge Technologies, Department of Information Systems, School of Mathematics and System Engineering.

32. **Sundberg, Klas (2009).** *Atlas Copcos strategi och styrning: Verktyg som ger guld***,** Licentiate Thesis, No. 48, Uppsala universitet, Företagsekonomiska institutionen.

33. **Mörndal, Marie (2009).** *"Hallå! Jag känner mig ensam här": En studie om studieovana studenters interaktion på ett webbaserat diskussionsforum,* Licentiate Thesis No. 113, Mälardalens högskola.

34. **Svensson, Martin (2010).** *Routines for Engagement: Emotions and Routines when Communicating through ICTs,* Linköping Studies in Science and Technology, Thesis No. 1444 LiU-TEK-LIC 2010:15.

35. **Mansour, Osama (2011).** *Share with Social Media: The Case of a Wiki,* Licentiate Thesis, School of Computer Science, Physics, and Mathematics, Linnaeus University, Växjö.

36. **Gullberg, Cecilia (2011).** *Puzzle or Mosaic? On Managerial Information Patterns*, Licentiate Thesis, Linköping University, Institute of Technology. Licentiate Thesis, ISSN 0280-7971; 1483.

37. **Kajtazi, Miranda (2011).** *An Exploration of Information Inadequacy: Instances that Cause the Lack of Needed Information*, Licentiate Thesis, School of Computer Science, Physics, and Mathematics, Linnaeus University, Växjö.

38. **Eklinder-Frick, Jens (2011).** *Building Bridges and Breaking Bonds: Aspects of Social Capital in a Regional Strategic Network*, Licentiate Thesis No 139, Mälardalen University.

39. **Hansen, Therese (2012).** *Internet or Traditional Stores: Identifying Influences on Consumers' Mobile Phone Purchases*, Licentiate Thesis No. 50, Department of Business Studies, Uppsala University.

40. **El-Mekawy, Mohamed (2012).** *From Societal to Organizational Culture: The Impact on Business-IT Alignment*, Licentiate Thesis, Department of Computer and Systems Sciences, Stockholm University.

41. **Hadjikhani, Annoch (2013).** *Expectations in the Internationalization Process: The case of two Swedish banks' foreign activities 1995-2010*, Licentiate Thesis, Mälardalen University.

42. **Salavati, Sadaf (2014).** *Novel Use of Mobile and Ubiquitous Technologies in Everyday Teaching and Learning Practices: A Complex Picture*, Licentiate Thesis, Linnaeus University.

43. **Williamsson, Ia (2014**). *On the Possibility of Improving Software Quality: By Improving Software Maintenance*, Licentiate Thesis, Linnaeus University.

44. **Lagin, Madelen (2015).** *Assumptions of Retail Price Strategy and Price Tactic Decisions*, Licentiate Thesis, Örebro University.

45. **Aasi, Parisa (2016).** *Organizational Culture and Structure Influence on Information Technology Governance*, Licentiate Thesis, Department of Computer and Systems Sciences, Stockholm University.

46. **Eriksson, Emelie (2017).** *Patterns of Corporate Visual Self-representation in Accounting Narratives*, Licentiate Thesis, Department of Management and Engineering (IEI), Linköping University.

47. **Golshan, Behrooz (2018).** *Digital Capability and Business Model Reconfiguration: A co-evolutionary perspectiv*e, Licentiate Thesis, Department of Informatics, Faculty of Technology, Linnaeus University.

48. **Svensson de Jong, Ilse (2018).** *Explorative study on Measurement of Innovation*, Licentiate Thesis, Department of Design Sciences, Faculty of Engineering, Lund University.

49. **Haglund, Lennart (2019).** *Affärsrelationer på den svenska elmarknaden*, Mälardalen University Press Licentitate Thesis, L-279, School of Business, Society and Engineering, Division of Marketing and Strategy, Mälardalen University**.**

50. **Panahi, Parisa (2019).** *The role of information technology in supporting management control systems for corporate social responsibility*, The Department of Business Administration, School of Business, Economics and Law, University of Gothenburg.

51. **Lindeberg, Fredrik (2019).** *Coordinating the Internet*, Licentiate Thesis, Department of Management and Engineering (IEI), Linköping University.

52. **Jonathan, Gideon Mekonnen (2020).** *Information Technology Alignment: The Role of Organisational Structure*, Licentiate Thesis, Department of Computer and Systems Sciences, Stockholm University.

53. **Hönel, Sebastian (2020).** *Efficient Automatic Change Detection in Software Maintenance and Evolutionary Processes*, Licentiate Thesis, Department of Computer Science and Media Technology, Linnaeus University.