Automated Software Engineering (2021) 28:3
https://doi.org/10.1007/s10515-021-00280-3

Investigation and prediction of open source software evolution using automated parameter mining for agent-based simulation

Daniel Honsel[1] · Verena Herbold[1] · Stephan Waack[1] · Jens Grabowski[1]

Received: 6 April 2020 / Accepted: 11 March 2021 / Published online: 14 May 2021
© The Author(s) 2021

Abstract

To guide software development, the estimation of the impact of decision making on the development process can be helpful in planning. For this estimation, often prediction models are used which can be learned from project data. In this paper, an approach for the usage of agent-based simulation for the prediction of software evolution trends is presented. The specialty of the proposed approach lies in the automated parameter estimation for the instantiation of project-specific simulation models. We want to assess how well a baseline model using average (commit) behavior of the agents (i.e., the developers) performs compared to models where different amount of project-specific data is fed into the simulation model. The approach involves the interplay between the mining framework and simulation framework. Parameters to be estimated include, e.g., file change probabilities of developers and the team constellation reflecting different developer roles. The structural evolution of software projects is observed using change coupling graphs based on common file changes. For the validation of simulation results, we compare empirical with simulated results. Our results showed that an average simulation model can mimic general project growth trends like the number of commits and files well and thus, can help project managers in, e.g., controlling the onboarding of developers. Besides, the simulated co-change evolution could be improved significantly using project-specific data.

✉ Verena Herbold
vherbold@cs.uni-goettingen.de

Daniel Honsel
dhonsel@cs.uni-goettingen.de

Stephan Waack
waack@informatik.uni-goettingen.de

Jens Grabowski
grabowski@informatik.uni-goettingen.de

[1] Institute of Computer Science, University of Göttingen, Göttingen, Germany

Keywords Software evolution · Agent-based simulation · Software repository mining · Change coupling networks · Open source software

1 Introduction

The analysis of software repositories can be used to detect and forecast software evolution trends. Topics of interest are manifold and include developer contribution behavior (Ben et al. 2013; Joblin et al. 2017; Lima et al. 2015), changes and growth (Robles et al. 2005; Shihab et al. 2012; Kim et al. 2008), software network evolution (Amrit and van Hillegersberg 2010; Bhattacharya et al. 2012), and the prediction of bugs (Wiese et al. 2015) and effort (Kocaguneli et al. 2011). The aim of investigating software evolution is to understand the underlying processes which drive and impact the evolution, e.g., the reason why certain changes are performed. The history of changes is stored in the Version Control System (VCS) and is available for analysis. In the case of Open Source Software (OSS), the data is usually stored in repositories hosted on collaborative coding sites like GitHub. Software evolution simulation approaches use a model of the system which is filled with knowledge by mining real software projects. Since the changes made by the developers of the software are responsible for its evolution, we build our model upon the developers and their change behavior. Agent-based Modeling and Simulation (ABMS) allows us to model software evolution from the perspective of the developers performing changes to the software entities, e.g., files, classes, or modules.

Simulating the actions of developers, can help to estimate the future maintenance effort spent in the project and, thus, aid in planning. In practice, it is hard to figure out simulation models which are valid for the whole application area (Sargent 2011). Using project-specific simulation models retrieved from empirical data can mitigate this problem, but may be not suitable for every type of software project. A close interaction between the mining step as parameter estimation and the simulation framework for the instantiation provides researchers with large testing possibilities. This establishes an important step in assessing the validity of software process simulations.

This work is based on the PhD thesis of the first author (Honsel 2019) and has the following contributions:

- The introduction of an automated parameter estimation approach including the development of a framework to instantiate simulation models for software evolution.
- An elaborated simulation model of software evolution which builds upon lessons learned from past studies (Honsel et al. 2014, 2015, 2016a).
- A case study including ten OSS projects comparing project-specific and general software evolution models as well as the evaluation of these for different points in time, i.e., short-term versus long-term prediction.

The remainder of this paper is structured as follows: We present the background of software evolution and agent-based simulation in Sect. 2. In addition, an overview on related work in the field is given in this section. In Sect. 3, we present a motivating example to illustrate possible applications of our work. Then, we describe our approach

of automated simulation parameter mining from OSS project data and the associated simulation process in detail in Sect. 4. In Sect. 5, we present our case study on the validation of different software evolution simulation models. Finally, we conclude the paper in Sect. 6 and give an outlook on future work.

2 Background and related work

In this section, we present the foundations of software evolution and how questions concerning software evolution can be analyzed. Then, we go into detail about agent-based simulation models and their ways of validation. In addition, we present the state of the art and common approaches from related work. For our work, we combine methods from different fields. Thus, it is important to present the basic ideas behind the methods used as well as how they interact.

2.1 Analysis and prediction of software evolution

The investigation of software evolution dates back to Lehman (1980) who first investigated program evolution within his work at IBM. As an outcome of this research, the laws of software evolution were formulated that try to describe the interplay between changes, related system growth and complexity, and need for program adaption, e.g., maintenance activities. These investigations induced increasing interest in the topic of software evolution research, which focuses on understanding the causes and impact of changes to the software. Entities involved in software evolution are the developers since the perform the changes to the software, software artifacts since they are changed, and bugs since they may be introduced due to changes.

Important research topics reach from developer contribution behavior (Lima et al. 2015; Bhattacharya et al. 2014), the nature of OSS and its evolution (Khondhu et al. 2013; Fernandez-Ramil et al. 2008; Alfayez et al. 2017), software changes (Shihab et al. 2012; Kim et al. 2008), or the causes, localization and prediction of bugs (Lamkanfi et al. 2010; Wiese et al. 2015; Rahman and Devanbu 2011).

We start our investigations from the inside of the software projects, i.e., examining the work performed by the developers. Their changes directly impact the growth of the software system. Dependencies between the different actors in software evolution like developers, software artifacts, and bugs, can be represented by networks. Following these strategies, we can analyze the evolution of the work of developers and related software growth as well as the evolution of structural properties of the software system. Respected data can be retrieved from VCSs, Issue Tracking Systems (ITSs), and mailing lists by software repository mining.

Mockus et al. (2002) presented an interesting case study of two major OSS projects: Apache and Mozilla. The authors aim to understand the nature of OSS software projects. For this, they analyze different aspects of the software evolution process, e.g., team structure, the role of core members, communication, and bug fixing behavior.

Developer behavior and roles

The work of developers, especially in OSS projects, depends on many factors like knowledge, experience (Bhattacharya et al. 2014), background, interests, and motivation (Li 2012). All of these can vary over the time. To measure the contributions performed by individual developers, no general method exist. Developers' contribution can be described as the entire list of activities a developer performs during the development of the software (Gousios et al. 2008). Several approaches try to express the contribution of a developer in a timespan by, e.g., the number of commits performed in this interval, the number of Lines Of Code (LOC) written (Yamauchi et al. 2018; Bird et al. 2011), or the number of feature requests solved.

To sort developers who behave similarly, developers can be classified by their role in the software project. A common approach for this task is the onion model (Crowston and Howison 2006) which assumes that 80% of the work performed in a software project is done by the top 20% of the developers. This leads to a structure where the top contributors present the core developers whereas the rest are peripheral developers. This core/periphery structure is utilized for many studies (Yu and Ramaswamy 2007; Amrit and van Hillegersberg 2010; Terceiro et al. 2010). Joblin et al. (2017) evaluated the findings from count based contribution measures and compare them with metrics retrieved from hierarchy networks. As a result, they found that count-based metrics deliver consistent results, but enriched with information from collaboration networks it can support research in this direction. Taking this into account, we propose a count-based model reflecting different contribution activities and different roles of developers.

Software changes and growth

Software changes to affected software entities, e.g., file, classes, or software modules (Turski 1996), impact the growth of the software system directly. The addition and deletions of files change the file count whereas modifications also change the LOC count (total number of LOC). Both are common measures for the size of the software. Godfrey and Tu (2000) showed that such growth behavior often follows sub-linear trends, but also other patterns are possible, e.g., super-linear growth patterns (Herraiz et al. 2006). In our work, we take the number of files (NOF) as measurement for the system size into account.

Changes to the software can have different purposes. Usually, a change set of files is changed for solving one or more tasks. These tasks can represent a feature addition, a bug fix, or other maintenance activities. Since developers also incidentally fix bugs, complete various tasks at once or commit work in progress, it is not easy to map developer activities to their intentions, i.e., tasks, exactly (Konopka and Navrat 2015). Several approaches exist to untangle these changes, e.g., Herzig and Zeller (2013); Hindle et al. (2009). An issue which is especially relevant for our work is to identify changes as bug fixing changes. To define the workload of different developer types, it is important to know how many commits and bug fixes they perform. A common approach is the SZZ algorithm (Kim et al. 2008) which does exactly this classification.

Since the SZZ algorithm has some disadvantages, e.g, many false positives, we use validated data by Herbold et al. (2019) for our work.

Software networks

A software network can be defined as a set of nodes N which can represent any kind of software entities, persons that are involved in the software development process, or bugs that affect the software. Software entities can also be considered on the function level, e.g., for constructing function call graphs. The relations between the different nodes are defined by a set of edges E which link two nodes $n_1, n_2 \in N$. Generally, networks can be directed or undirected depending on the kind of relationship. Here, we only consider undirected networks. Software networks can represent structural properties of the software under investigation, e.g., call graphs (Bhattacharya et al. 2012) and inheritance graphs, or collaborative properties, e.g., developer co-editing (Caglayan et al. 2013), or bug assignments (Bhattacharya et al. 2012). Interesting research topics using software networks in software engineering reach from the detection of core developers (Huang and Liu 2005) over communication activities in OSS projects (Bird et al. 2006) to bug prediction (Meneely et al. 2008).

In social network analysis, a common task is to detect communities within the network. A community is defined as "…groups of vertices which probably share common properties and play similar roles within the graph" (Fortunato 2010). In software networks, communities can represent different entities, e.g., packages, related test files, or teams of developers.

For finding evolutionary patterns in the growth of software networks, we trace some network metrics over the time. To represent structural as well as semantically related software entities, we take change coupling graphs into account (Ball et al. 1997; D'Ambros et al. 2009). Change coupling is defined as "…the implicit relationship between two or more software artifacts that have been observed to frequently change together during the evolution of a software system" (D'Ambros et al. 2009). The analysis of change coupling evolution is often used for change recommendation since software entities that have often been changed together in the past are likely to be changed together in the future, or to detect design issues (Gall et al. 1998). Apart from that, the evolution of itself as well as its impact, e.g., on defects, is of interest (D'Ambros et al. 2009). We also consider communities of the change coupling graph as well as respect the change coupling structure in the file change strategy of developers.

2.2 Agent-based simulation

Agent-based simulation can model complex scenarios with rather simple rules. The main challenge in ABMS is the model building process. This includes the description of the behavior of the agents, which can be implemented rule-based. The model building process involves finding a balance between assumptions you make on the model and finding suitable changeable parameters. Agents are individuals with an autonomous behavior which can be adapted (Macal and North 2005), i.e., the agents can learn over

the time. Moreover, agents can interact which each other. Agents can be active via performing actions on their own as well as passive by just changing their state and attributes. Applications of ABMS show a wide range from business market analysis to health related issues, e.g., the humane immune system (Macal and North 2006).

In software engineering, ABMS is seldom used. Most similar to our approach is the work of Smith et al. (2006) who propose an agent-based simulation model for software evolution with developers, requirements, and modules as active agents. In their work, they use a grid-based representation of the software project where the agents can move around in their neighborhood each simulation round. Hence, the developers can only work on nearby modules. Moreover, they can leave the project by moving outside of the grid. Additionally, developers can leave the project according to a motivation factor. Possible developer activities are the creation of new modules on requirements, refactoring modules, and develop modules. For validation, the authors compare empirical trends like the growth and the complexity of the software system with simulated trends like we do in our research. Compared to Smith et al. (2006), we reflect also structural properties of the software, the evolution of bugs, and developer roles. Moreover, we use networks instead of a grid as a topology which makes the simulation more flexible and able to represent different perspectives on the systems' evolution, e.g., relationships between developers and evolution of logical coupling of the software.

Garcia-Garcia et al. (2020) published a recent literature review on Software Process Simulation Modeling. They identified 36 papers as state of the art from 2013 and 2019. Within their study, agent-based simulation and Discrete-Event Simulation are the most used paradigms with 25% each. Still, most of the work listed there focus on a defined environment, e.g., process model, or a given task, e.g., predict the risk of the project to fail or the training of junior system engineers and project managers. In our work, we use ABMS for more general software evolution scenarios and trend prediction.

An interesting study related to our work is presented by Ali et al. (2018). The authors also investigate whether an agent-based model of software evolution can reflect real world processes. Different to our work, they reworked an existing System Dynamics simulation into an agent-based framework and checked the appropriateness for this purpose. For describing the complexity of a software project, they use Actor Network Theory which include different actors and their interrelations, e.g, project managers, developers, or mutable tools with the overall aim to measure the health of the system. Furthermore, they performed an evaluation of the effect of attitude changes, e.g., by the project manager. Their evaluation is based on an experts opinion instead of real-world data, since their model requires a lot of project information to be instantiated as a real software project.

2.3 Validation of simulation models

For the validation of simulation models, several approaches exist depending on the purpose of the simulation. Sargent (2011) pointed out that simulation models should be evaluated with respect to the answers the model is targeted to answer. Therefore, one has to specify needed output variables as well as the extent of accuracy needed

to answer the questions adequately. Usually, tests are performed until the model is valid for a sufficient set of experiments. The following aspects should be considered: if model assumptions are valid, if it is well designed towards its intended purpose, and checking if the system is implemented right. Furthermore, it is important to check the output behavior of the simulation model (Sargent 2011). One of such approaches is the comparison of empirical with simulated data. To support the internal validity, it is also common to have several runs of a simulation to mitigate model variability. An advantage of the approach presented in this paper is the facilitation of predictive validation, since data can be taken as input for selected points in time. Thus, the prediction results can be easily compared through the collaboration with the mining framework.

3 Motivating scenario

Generally, the motivation of our work is two-fold. First, we aim to provide decision support for software project managers by estimating the impact of different parameter sets, i.e., project settings. Second, we want to support researchers who are interested in the simulation of software evolution by providing a mining and simulation framework which are closely working together. By this, different outputs are available for analysis and for a smarter validation of simulation models. In the following, we only provide an example scenario for the first layer, since the second layer emerges from the paper.

Scenario From the starting point of a project manager, it is an important task to estimate the future effort spent by the developers. Usually, this is based on expertise. This is not an easy task since OSS development is an undetermined environment. Our idea is to use a simulation model based on previous evolution of the software. We assume that such a project specific simulation model can yield realistic predictions. For comparison, we assume that an average behavior taken from a set of OSS projects is not adequate to forecast the ongoing in a software project sufficiently. As a result of the comparison it gets clear whether an average model is powerful enough to predict software evolution trends and how much empirical data can improve simulation results.

A concrete use case considers project managers monitoring the onboarding process: The progress in many OSS projects hosted on, e.g., GitHub is controlled by the incoming pull requests. Usually, every pull request needs to be reviewed by a core developer and, thus, the number of commits is limited by the number of core developers (committer). An active core of the project is essential to keep the project healthy. Hence, it is an important task for the project manager to determine the number of developers to onboard.

Figure 1 shows the project manager thinking about an appropriate project setting, e.g., to find out the appropriate number of new developers to onboard. He decides to use the proposed simulation framework to test the impact of different factors. Changeable parameters are the number of developers he plans with, the expected size of the software system, and the development time in days. Then, the simulation runs and predicts the estimated trends, e.g., the workload performed by the developers. It is possible to adapt the parameter set if the simulation output is not in line with his conception. This establishes a feedback loop for project managers and developers.

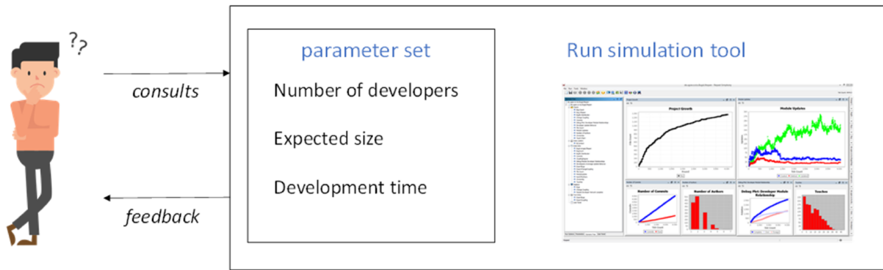


Fig. 1 Usage scenario of proposed simulation framework

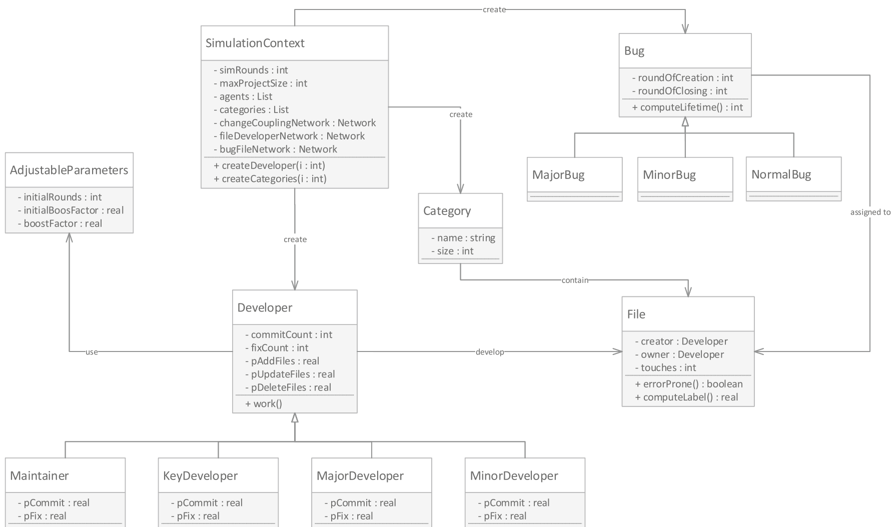


Fig. 2 Agent-based simulation model

4 Methodology

In this section, we present our approach in detail. Since our goal is to analyze the evolution of OSS projects using agent-based simulation, we start with the description of our simulation model. We proceed to describe how we mine software repositories to automatically estimate required parameters to instantiate the simulation model followed by a description of the mining framework. Finally, we describe how the tools presented here can be applied to almost any open source project.

4.1 Simulation model

The agent-based simulation model used for this evaluation is depicted in Fig. 2. The main agents as well as their interaction is described below.

The *SimulationContext* represents the initial class of the simulation model. At simulation start-up, it creates a given number of developers and is initialized with

project-specific parameters such as the maximum project size and the number of rounds to simulate. Furthermore, the SimulationContext knows all instantiated agents as well as all networks, representing dependencies between the agents, at runtime. The simulation is round-based and one round in the simulation represents one day in real life. Every turn each developer has the opportunity to work. Whether she makes a contribution depends on her role and the associated probabilities.

The *Developer* is considered as active agent because its development work adds, deletes, and updates files. This contribution makes the project evolve over time. Development work means to apply a commit. How often a developer performs a commit and how many files are affected per commit depends on the probabilities for the specific developer type based on the mining process. The developer types *KeyDeveloper*, *MajorDeveloper*, and *MinorDeveloper* mainly differ in their commit frequency. The *Maintainer* is similar to the *KeyDeveloper*, only with a particularly high number of bugfixes. How the classification works in detail is described in the mining process.

Categories represent folders or packages of the software in which files are grouped together in a logical context. When a file is created, it will be assigned to a category with a certain probability depending on the size of the category. This allows us to simulate the clusters of the change coupling more realistically.

To make a statement about the quality of the software under simulation, we use the number of *Bugs* assigned to the *Files*. For this, bugs are weighted according to their type. A bug is generated by the SimulationContext according to the creation probability of the specific bug type. After the instantiation the bug will be assigned to an almost randomly selected file. According to D'Ambros et al. (2009), a property that makes a possible assignment of a bug to a file more likely is the coupling degree of the file. We model this by computing the error proneness based on the coupling degree, i.e., the number of links to other files, of the file in the change coupling network which is introduced below. This behavior simulates the occurrence of bugs managed in an issue tracking system and created by users, testers, or developers. Each commit a developer can fix a bug with a certain probability depending on the type of the developer.

Dependency networks

To model a software system more accurately, dependencies between involved agents have to be described. For this we use networks where the nodes represent the agents and they can be connected by weighted edges. The following three networks are defined for our simulation model.

File—developer network This network represents the dependency between a *File* and a *Developer*. The first edge connected to a file is created when a developer creates the file. Further edges are created when a developer modifies a file that has not previously connected to the developer. If a developer modifies a file and there already exists an edge between them, then the weight of this edge will be increased. The state of a file can change with every commit. An example of the file—developer network is depicted in Fig. 3.

This network provides the the following properties of a file: the *owner* as the developer that changed the file at most; the *number of authors* represented by the

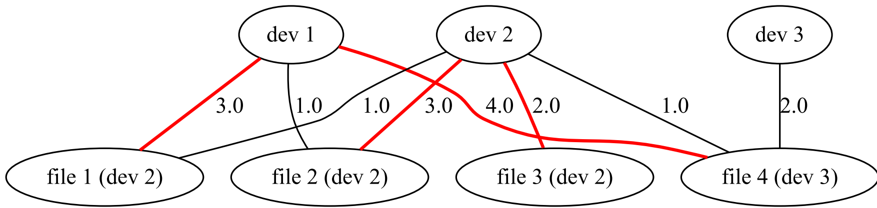


Fig. 3 Example for a file—developer network. Touches are represented as weight of an edge. The red marked edges represent the owner of a file. Behind the filename is named in brackets the creator of the file (color figure online)

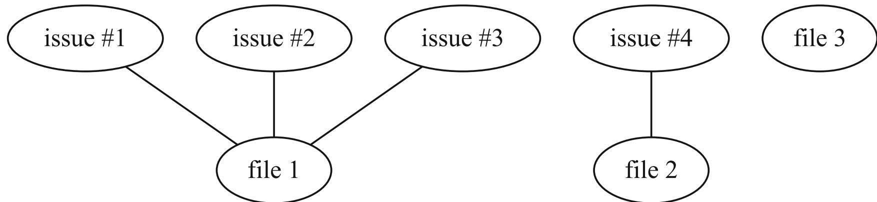
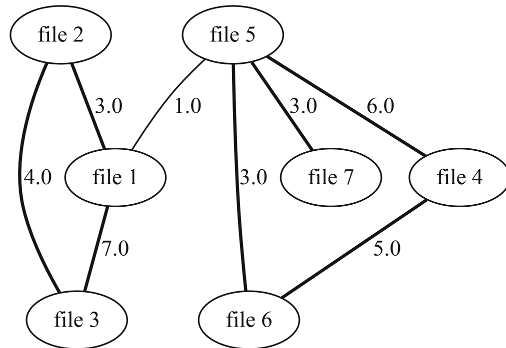


Fig. 4 Example for a bug—file network. Any created issue is assigned to an already existing file

Fig. 5 Example for a change coupling network. The weights of the edges represent how many times files are changed together in one commit. Bold printed edges represent the membership in a cluster



degree of the file in this network; the *number of touches* calculated by summing up the weights of all connected edges.

Bug—file network After a `Bug` is created and the `File` to assign is selected, both agents will be linked by an edge in this network.

The edge contains information whether a bug is closed or not. Thus, the bug does not have to be deleted and we can reopen it, if required. An example for a bug—file network is shown in Fig. 4.

Change coupling network This network describes dependencies between files that are changed several times together in a commit. In Fig. 5 an example for a change coupling network is presented.

This network provides the change coupling degree of a file. This is used, for example, for the determination of error prone files (D'Ambros et al. 2009). The change coupling network generates clusters of files that are changed often together. Accord-

ing to Ball et al. (1997), files of one cluster are semantically related. Therefore, this network is the most precise representation of the software under simulation.

Software changes and growth

A commit can consist of several file changes which can be additions, deletions or modifications. The growth of the software system as measured in the number of files (NOF) results directly from these changes, i.e., the file additions and file deletions. Depending on the role of the developer, it is determined whether the developer works in this simulation round and if so, how many file actions she performs. A model assumption behind these probabilities is that the file actions are geometrically distributed (Honsel et al. 2014). A geometric distribution counts the number of trials before the first success. In our scenario, this means that necessary adoptions are modeled as failure and a success is that the software is in a satisfactory state, thus, no adaption is needed. This can be expressed by $P(action) = (1-p)^k p$ with p the probability for each $action = \{create, update, delete\}$ and k the number of attempts. In this version of the geometric distribution, we have the expectation value $E = \frac{1-p}{p}$ and, thus, $p = \frac{1}{E+1}$.

To instantiate the simulation model, the parameters gathered by our mining framework are used. If the mined parameters are not suitable to simulate growth realistically, then we can use the adjustable parameters to tweak the developers contribution behavior. This is particularly useful when there is strong growth in the initial phase of the project.

Simulation framework

For simulation purposes we are using the agent-based modeling and simulation framework Repast Simphony (North et al. 2013). It provides a Graphical User Interface (GUI) to control and parameterize the simulation at runtime as well as time series or histogram charts of desired properties during the simulation.

4.2 Mining software repositories

In our approach, we use real data as input for the simulation to gain realistic results. This data stems from the history of software projects stored in VCSs, ITSs, and mailing lists. In this section, we describe how we learn aspects required for an agent-based simulation model of software evolution from this data. For the comparison of empirical and simulated behavior, we need a set of metrics, which are introduced in the following sections.

Developer behavior and roles

An important point in designing an agent-based simulation model for software evolution are the changes to the software. For this, it is indispensable to get a deeper understanding on the way developers work since they are responsible for the changes.

Fig. 6 Developer classification own approach versus Onion Model

To grasp the structure of the software team constellation, developers are classified into roles depending on their commit behavior. For this, we take heuristics about commits and bug fixing commits into account. A bug fixing commit is a commit labeled as bug fix as explained in Herbold et al. (2019).

For the developer classification, we tested different thresholds for the separation into the following three types: core, major, and minor developer. We identified the following classification as applicable for many software projects: A developer performing over 30% of the overall commits is defined as core developer, with over 2% of all commits as major developer, and all developers with less commits are classified into minor developers. A core developer can have the additional role of a maintainer, if she performs over 15% of all bug fixes (Herbold 2019). To demonstrate the applicability of this classification, we illustrate our approach for an existing open source project. For this, we calculate all role separation thresholds both using our approach and the onion model (Crowston and Howison 2006) for the OSS project Zookeeper. In Zookeeper, 87 developers were active in the observed timespan. Applying the thresholds mentioned above, our approach identifies one core developer, ten major developers, and 76 minor developers. Using the 80% percentile, the onion model generates 20 core developers and 67 peripheral developers.

Figure 6 shows all developers of Zookeeper sorted by their total number of commits. Our approach is colored for the different roles whereas the onion model threshold is presented as blue line at 17 commits. Thus, all developers with at least 17 commits are classified into core developers and the rest as peripheral developers. From this, we construe that for projects with a long tail, i.e., many tiny developers, our model is more suitable because the diversity of developer workload within the groups is quite large using the onion model.

We observe this behavior for many OSS projects. Still, for projects where the number of commits (NOC) are distributed more equally, the onion model splits the

developers similar to our approach. Then, core and major developers conform to core developers and minor developers conform to peripheral developers.

We trace the evolution of the NOC performed by each developer regarding the commit behavior over the time. The number of bug fixing commits are also taken into account for modeling the probabilities of the different developer roles to fix a bug within a commit.

Software changes and growth

As mentioned above, the commit behavior of the developers follows a geometric distribution. In the used version of the geometric distribution, we have the expectation value $E = \frac{1-p}{p}$ and, thus, $p = \frac{1}{E+1}$. Hence, for the estimation of parameters, we take all observations from the version history belonging to the same developer role and action and derive the probabilities p from the expectation value of the corresponding observation sequence.

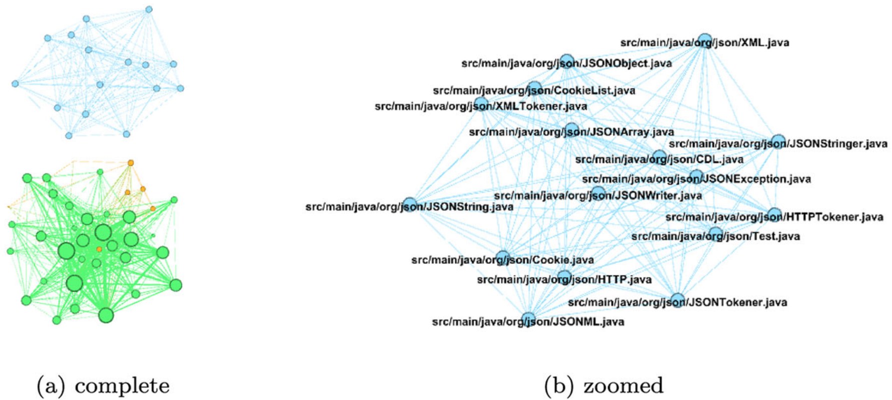
The probability whether a change introduces a new bug is based on the bug fixing probabilities of the different developer types. For this, the ratio of bug fixing commits and other development commits is calculated and taken as average for each developer type. The bugs are categorized into major, normal, and minor bugs mirroring the occurrences retrieved from the ITS. Major bugs cover major bugs and worse, e.g., critical or crash. The bugs evolution and lifetime is evaluated in our former work (Honsel et al. 2016a) and, thus, not in this paper.

Software networks

We consider different kinds of software networks for the simulation of software evolution: developer-file networks for the representation of collaboration, file-bug networks for the distribution of bugs among the files, and change coupling networks for interdependencies between the files. In this work, we concentrate on the change coupling networks which we select to investigate the evolution of the structure of software projects. In former studies (Honsel et al. 2015, 2016a), we already showed the applicability of change coupling graphs for the simulation of software evolution, but the mining part was done manually and, thus, the evaluation could not be performed on a big scale. As defined in Sect. 2.1, a change coupling network consists of a set of nodes $N = f_1, \ldots, f_n$ including all files f_i, $i \in \{1, \ldots, n\}$ and n the number of files that have been changed together with another file in one commit at least twice. An edge is the linkage of two nodes based on common changes in the commit history. Each further co-change increases the weight of the edge by 1.

For a visual representation of these networks, we use the tool Gephi (Bastian et al. 2009) which supports network analysis and visualization. Besides the number of nodes and edges over time, Gephi provides several statistics and network metrics. We use the following for our analysis:

- *Network modularity* The modularity of a graph $mod(G)$ is an indicator of how good the graph can be divided into clusters. Clustering based on the modularity uses a quality function to grasp the goodness of partitions (Fortunato 2010). The



(a) complete (b) zoomed

Fig. 7 Change coupling graph of Giraph after the first year of development

actual graph is compared with the so-called null model, that preserves the structure of the graph except for any community structure.

- *Network density* The network density sets the number of possible ties in a network in relation to the actual ties. Thus, a network largely connected is dense.

Since this work aims to assess more the distribution and the overall connectivity of files instead of ranking individual files, we neglect the node degree or other network metrics indicating the importance of single nodes.

For detecting clusters within the change coupling graphs, we use the clustering method based on the modularity function provided by Gephi. Gephi uses a greedy optimization method to optimize the modularity (Blondel et al. 2008). In the change coupling graph, these clusters represent logically related software entities, e.g., all files implementing the GUI of a software product.

We present an example for a change coupling network in Fig. 7. Figure 7a shows the full change coupling graph of Giraph. There, two large and one smaller communities can be obtained. A closer look at the blue community (Fig. 7b) reveals that it is a JSON package.

Usually, the used algorithm produces a structure similar to the actual packages, but it is possible that some packages are clustered together due to strong interdependencies or, vice versa, that packages are split into smaller ones, if they are not often changed together or rarely changed at all.

In our software networks, also additional information like the owner of the file is stored. The ownership is identified by the proportion of touches to the the file, i.e., the developer who performed most changes to it owns it. In the simulation, the files the developer changes are selected randomly for the first file. If the commit includes more than one file change, than the selection of further files depends on factors like ownership, the category and former changed files by the developer.

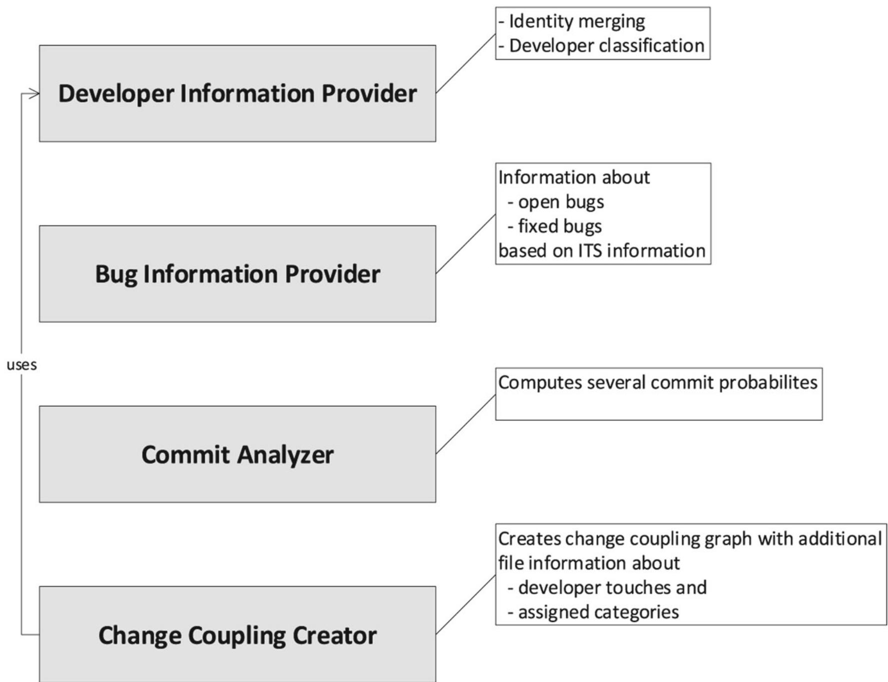


Fig. 8 Components of the overall process of automated parameter estimation

4.3 Mining framework

To gather required parameters for the instantiation of our simulation model, the following mining framework is proposed. Our mining framework requires data that is created by SmartSHARK (Trautsch et al. 2018). SmartSHARK contains a set of different tools that collect data from VCS, ITS, and mailing lists. All collected data is stored in a MongoDB. To generate the simulation parameters, the following four components, depicted in Fig. 8, are essential.

Developer information provider This component collects all developers that are authors of at least one commit of the analyzed project. Afterwards, two tasks are performed. First, the identities of the developers are merged using an adapted identity merging algorithm based on Goeminne and Mens (2013). This is necessary, for example, if one and the same developer uses different email addresses. Second, developers are classified into different types according to the simulation model.

Bug information provider This component provides information about the number of bugs that are created and fixed per year. Furthermore, all bug priorities of the ITS are mapped to the priorities of our simulation model.

Commit analyzer This component investigates all commits of the analyzed project and computes required probabilities to model the contribution behavior of the developer types. This means, the number of updated, added, and deleted files

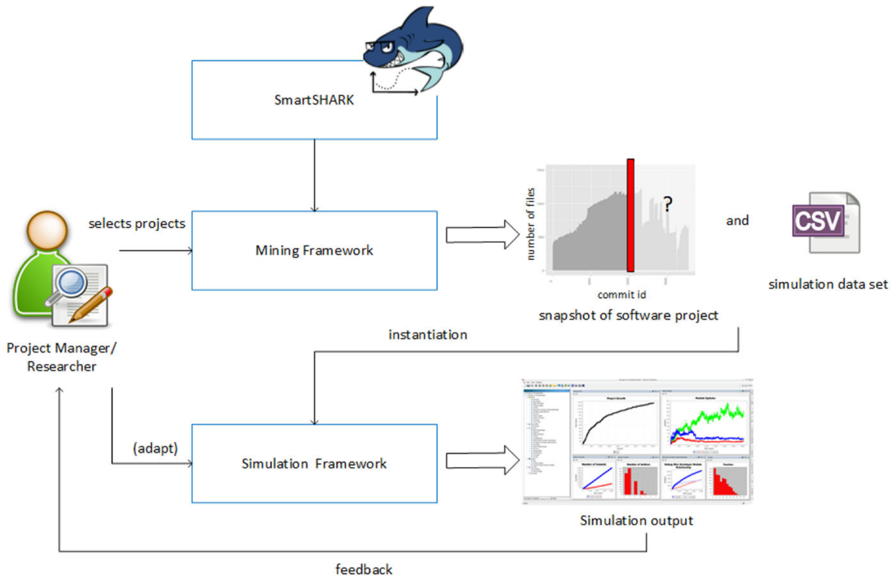


Fig. 9 Overall process of parameter mining and simulation of software evolution

per commit as well as the number of commits by type are analyzed to calculate the geometric distribution probabilities. Currently, we use two commit types: an general commit type and a bugfix. The commit classification is based on manually validated data by Herbold et al. (2019) is provided by SmartSHARK.

Change coupling creator This component creates the change coupling graph of the analyzed project based on the revision history for every year or any selected point in time. The change coupling graph is extended with additional file information about developer touches and assigned categories. This information is used to instantiate the simulation model at any given point in time.

The parameters generated by the first three components are exported as one JSON file. A parameter set is required for every instantiation of the simulation model. The change coupling graph is exported using the DOT format and will be used to instantiate the simulation based on a snapshot of the analyzed software at any desired point in time.

4.4 Overall process

Our approach combines the areas of software repository mining used for the automated estimation of simulation parameters and agent-based simulation used to simulate software evolution. The overall approach is depicted in Fig. 9.

This process is described in more detail below by explaining the individual steps required to analyze any project. A more technical description of these steps can be found in the general workflow[1] and in the documentation of the used tools.

[1] https://github.com/dhonsel/SimSE/blob/main/docs/workflow.md.

1. Create a MongoDB with all required data to execute our parameter estimation tool. For this, the desired open source project should be analyzed with SmartSHARK.[2]
2. Create simulation parameters and change coupling graphs with additional information to instantiate the simulation model at any year. For this, the before created MongoDB is required by the automated parameter estimation tool (Honsel 2020a).
3. Parameterize the simulation model (Honsel 2020b) with the parameters just obtained. Then, the simulation can be executed.
4. To improve the simulation results or to compare two different simulation runs, the runtime parameters can be changed before each simulation run as described in the documentation. Furthermore it is possible to start the simulation on the basis of real data on a desired year.
5. Evaluate the simulation results. Based on the research questions in mind, the evaluation method is chosen. For the comparison between different simulation runs as well as the assessment of the quality of simulation results, e.g., error measures can be used. It is also possible to train models on the empirical data and compare these with simulated results. An example for such an evaluation script is the replication kit belonging to this paper (Herbold 2020) which compares different simulation models gained from empirical project data.

This approach can be used to analyze several aspects of software evolution depending on your point of view. For researchers, the different snapshots of software projects can be used for, e.g., the analysis of software evolution trends. For project managers, simulation results can reveal important insights and, thus, can aid in decision making for software project planning.

Furthermore, the runtime parameters allow a project manager to run through different project runs using a kind of feedback loop to improve the simulation result.

5 Evaluation

In contrast to former studies, we can perform a larger evaluation of the proposed simulation model since the parameter estimation does not need to be done manually anymore. We are especially interested in the quality of a basic model of software evolution compared to models fed with different amounts of project-specific information. In previous studies, we always initialized the simulation model with knowledge from one project or the average behavior of a small set of OSS projects. The new mining tool allows us to build project-specific simulation models. Overall, we aim to answer the following research questions (RQs):

RQ1 Is a project-specific simulation of software evolution better than a simulation with average project behavior?

We aim to assess the impact of a project-specific parameter estimation. One aspect that did not become clear during our previous studies is if parameter tuning for the behavior of agents, i.e., the developers, can yield better results. Besides, there exists no evidence in literature (Herbold 2019) to which extent the usage of empirical data

[2] https://github.com/smartshark.

for parameter estimation is beneficial for the simulated results. This establishes an extra motivation to perform different tests on this aspect.

We derive the following hypothesis belonging to this research question:

H1 We assume that a project-wise simulation model can yield significant better results in terms of the NOC and the NOF than the baseline model.

The second research question aims the answer whether a simulation model that is initialized after one third of the project can improve simulation results.

RQ2 How accurate is a long-term prediction of software evolution trends compared to a short-term prediction?

H2 We expect that a snapshot-based simulation model can yield significant better results in terms of the density and the modularity of the change coupling graph than the project-wise simulation.

Taking a real snapshot for initialization could improve the change coupling metrics, since in contrast to the other initialization setups, the structure of the change coupling graphs is not solely controlled by the change behavior and change history of files.

5.1 Design and objectives

Since the proposed mining framework builds upon SmartSHARK, we evaluate our approach using a subset of Apache projects contained in the SmartSHARK database. We have selected 10 actively developed projects that differ in size, duration, and contribution. Details of the projects are depicted in Table 1.

For our experiments we selected the following ten Apache[3] projects:

- *Commons Collections*—an extension of the Java Collections Framework;
- *Commons IO*—a collection of I/O utilities;
- *Commons Lang*—additional functionality for classes in java.lang;
- *DeltaSpike*—a collection of portable CDI extensions;
- *Directory Fortress core*—a standards-based authorization system using an LDAP backend;
- *Giraph*—an iterative graph processing system;
- *Gora*—a framework that provides an in-memory data model and persistence for big data;
- *JSPWiki*—a WikiWiki engine;
- *Nutch*—a production ready web crawler;
- *Zookeeper*—a centralized service for distributed systems.

The team constellation is determined using our proposed commit threshold based approach (as described in Sect. 4.2). We target to achieve a high range in the system size and team constellation within the selected projects, i.e., we consider a projects' team size from ten to 139 developers.

For each examined software project, we instantiate the different simulation models with project-specific parameters required for our comparison. Thereby, we have three

[3] https://www.apache.org/.

Table 1 Overview of projects

Project	Commits	Max(Files)	Developers $(core\|major\|minor)$	Duration in months
Commons-collections	3358	1599	(1|7|50)	211
Commons-io	2259	453	(0|9|49)	202
Commons-lang	5782	659	(0|9|130)	195
Deltaspike	2310	1774	(1*|5|46)	80
Directory fortress-core	1609	828	(1*|2|7)	91
Giraph	1117	1333	(1*|10|28)	97
Gora	1327	477	(1*|11|24)	103
Jspwiki	8784	1682	(1*|5|9)	210
Nutch	3507	892	(0|12|61)	162
Zookeeper	2940	656	(1*|10|76)	128

The * marks core developers which are also maintainers of the project

different settings for each project included in the case study: the project-specific initialization, where all parameters are estimated individually, an average model, where the average over the work of developers per commit is used, and a simulation fed with project information from the first third of the observed project duration.

For ground truth, we utilize the empirical project data retrieved from our mining framework. The mining process covers different aspects of software evolution. This establishes the first step of our investigation since all models are parametrized based on empirical data. Thus, the amount of developers of the obtained developer types is determined including the effort spent by the developers when they perform a commit, i.e., the addition, modification, and deletion probabilities of files. The amount of changes contained in a single commit can vary a lot among the projects due to different coding styles, commit behavior of involved developers, and also the size of the changes. Besides the file change distribution and developer roles, we consider the related growth in NOC and NOF which can be easily measured analyzing the VCS. All metrics are calculated and compared yearly.

For the structural evolution of the project, we build the corresponding change coupling networks for each year, i.e., the representation of the relationship between co-changed files. Here, we trace the modularity as well as the density over the years as described in Sect. 4.2. The growth of the graph is based on the behavior of the developers as well as the package structure of the project in the average as well as in the project-specific simulation. For the snapshot, the mined change coupling graph after one third of the project is taken as input for the following remaining simulation period. We chose one third of the project as offset for the snapshot simulation because we assume this to be an sufficient timespan such that the initial phase is over, but that the project is not yet in a mature state which could have the potential effect of overfitting. Besides, a sufficient amount of remaining years is required to establish a fair comparison of simulation results. For the comparison of the resulting change coupling networks, we compare the network modularity and the network density evolution.

In summary, we have for each project and each metric four observation sequences: the empirical trend, the average simulation initialized with the average commit behavior over all projects, the project-specific simulation, and the snapshot simulation starting after one third of the software project.

5.2 Evaluation measures

First, we compare the prediction error introduced by the average simulation model with the error occurring when using the project-specific simulation model for the ten projects included in this case study to evaluate $RQ1$. The error is calculated considering the deviation to the empirical trend. For answering $RQ2$, we compare the observation sequences produced by the project-specific simulation with the sequence generated by the snapshot model. Here, the comparison starts the first year after the offset which is used as input for the snapshot, i.e., after the first third of the projects' current lifespan. For both test series we follow the same evaluation scheme:

1. *Calculate MAE and RMSE*
   To measure the error of the prediction, i.e., in our case the simulation results, compared with the empirical observation values, we use the Mean Absolute Error

(MAE) and the Root Mean Squared Error (RMSE) (Willmott 1982). Both are commonly used to evaluate prediction models. Both, MAE and RMSE, measure the average dimension of a set of prediction errors, but due to the square root in the case of RMSE large errors are penalized more.

2. *Test Normality*
The choice of the statistical tests for comparing two populations depends on their distributions. Thus, we test the normality of the two observation sequences using the Shapiro-Wilk test (Shapiro and Wilk 1965). The null hypothesis for the Shapiro-Wilk test is that the population is normally distributed. This means, that for a single test a significance level of $\alpha > 0.05$ indicates a normal distribution. To counteract incorrectly rejected null hypothesis occurring when testing multiple hypothesis, we apply Bonferroni correction (Shaffer 1995) adjusting $\hat{\alpha} = \frac{\alpha}{n}$ with n the number of Shapiro-Wilk tests. In our setup, we compare the MAE and the RMSE for the project-specific and the average simulation for the four metrics (16 tests) in the first test series. We do the same for the second test series with the snapshot based simulation and the project-specific simulation initialized with a snapshot. Hence, we have a significance level of $\frac{\alpha}{32} = 0.00156$ for normality testing. Since we identified nine out of the 32 populations as not normally distributed, we based the choice of the following tests on this observation.
3. *Assess Difference between Populations*
For populations where at least one is not normally distributed, the Wilcoxon Signed Rank Test (Wilcoxon 1945) can be applied to test whether the difference between the observations is significantly high. The Wilcoxon test compares two populations. We perform the test 16 times. Applying Bonferroni correction, this implies a significance level of $\hat{\alpha} = \frac{\alpha}{16} = 0.0031$. Since not all observation sequences are normally distributed, we report the median values as main trend of the data.

5.3 Results

In this section, we present the results of our case study and answer our research questions. Before we go into detail about the analysis performed for the two research questions, we present some interesting mining results about the evolution of the selected software projects. All used data is published in a replication kit (Herbold 2020).

Growth trends and network evolution

With our simulation model in combination with our mining framework, we are able to reproduce different growth trends. As an example for common growth trends, we plot the growth measured in the number of files for the three open source projects Commons-io, Gora, and Zookeeper. These projects were selected because each of them has a different growth trend. The results are depicted in Fig. 10.

The examples show an approximately sub-linear growth in the case of Zookeeper, an approximately linear growth trend for Commons-io, and an approximately super-

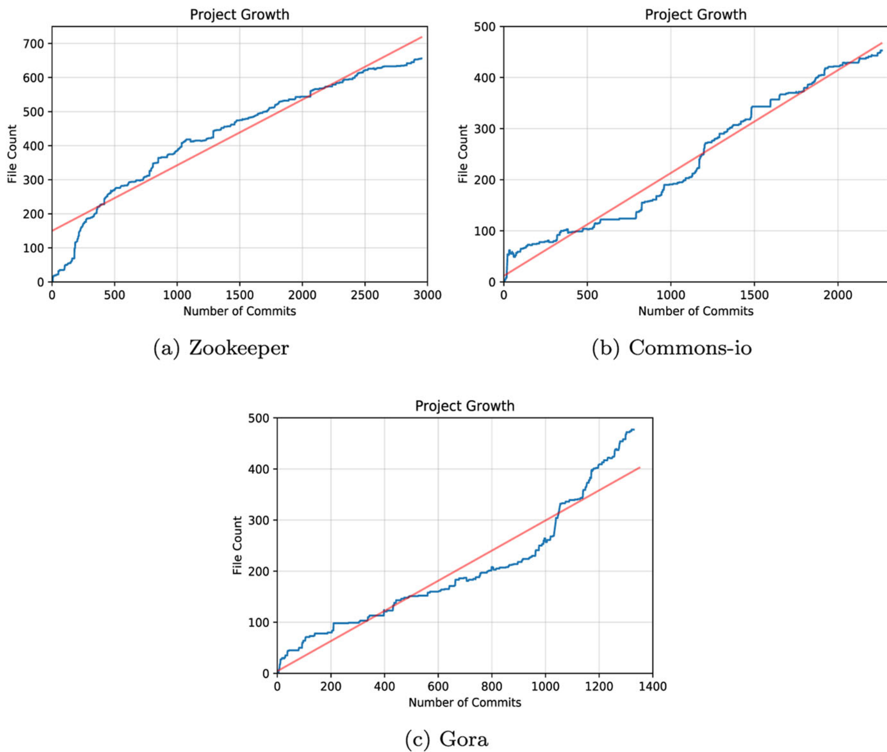


Fig. 10 Different growth trends of real software projects. The real growth is represented by the blue line and a linear fitted function by the red line (color figure online)

linear growth for the project Gora. According to Paulson et al. (2004), all growth trends can be fitted with a linear function.

For the evolution of the change coupling networks, we observed that the network density in the selected projects is always very low, i.e., lower than 0.25 as usual in larger networks and over the time. In contrast, the modularity of change coupling networks is rather high, i.e., between 0.5 and 0.8 in a mature state of the project (in the beginning also lower values are possible).

We exemplify the evolution of change coupling graphs in Fig. 11 and in Fig. 12. The graphs are presented for every two years of the projects lifespan. All graphs are colored based on modularity clustering (Blondel et al. 2008). The first graph shows the evolution of file dependencies of Zookeeper. There, a bigger, dense subgraph emerges over the time with some smaller clusters around. This behavior can be often observed for this kind of graphs. In our study, almost all projects showed this behavior. This means that huge parts of the software are interrelated. The evolution of Commons Lang is depicted in Fig. 12. This example shows an alternative behavior. Here, the graph evolves similar in the beginning, but the size of the clusters is more balanced at the end of the project. From the software engineering point of view such a development

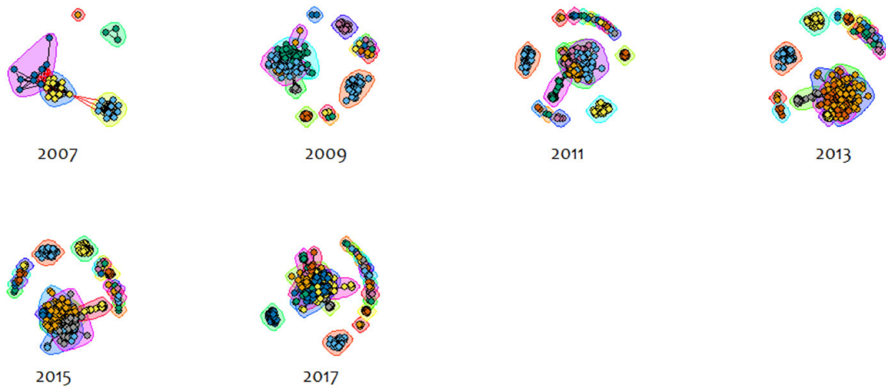


Fig. 11 Evolution of the change coupling network of Zookeeper

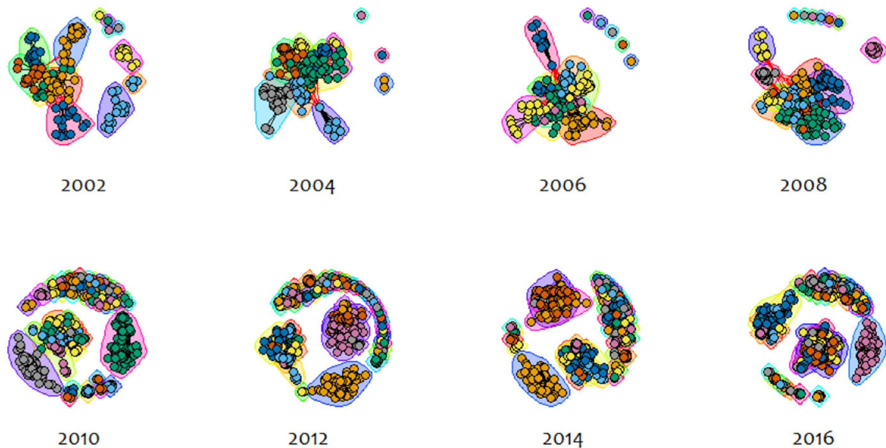


Fig. 12 Evolution of the change coupling network of Commons Lang

is more desirable because file changes may not have so many dependencies entailing further changes to other parts of the software.

The above observations can be explained considering the related evolution of the modularity metric. From Fig. 13, that shows the evolution of the change coupling modularity for Zookeeper for all simulation modes compared to the empirical trend, we gather that the snapshot simulation can increase the proximity to the real evolutionary trend. Compared with the same plot for Commons Lang in Fig. 14, we see that this does not have to be the case. Additionally, the values for Commons Lang are higher than for Zookeeper with a rise after half of the projects duration. This rise is also visible in Fig. 12 comparing the change of the graph structure from the year 2008 to the year 2010.

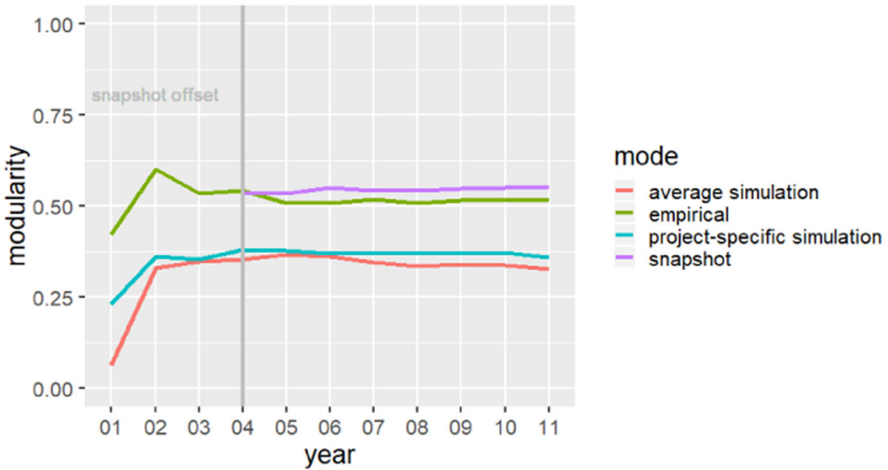


Fig. 13 Modularity of Zookeeper

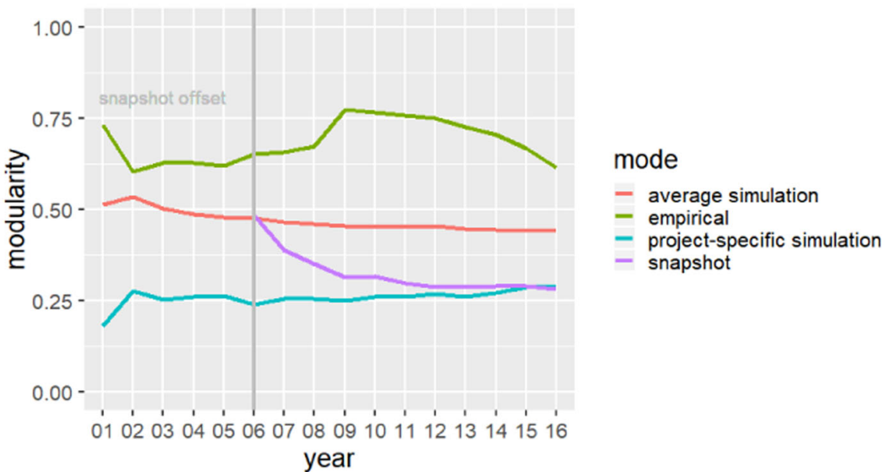


Fig. 14 Modularity of Commons Lang

RQ1 **Is a project-specific simulation of software evolution better than a simulation with average project behavior?**

For answering $RQ1$, we compare two different simulation setups. The first setup initializes our proposed simulation model for software evolution with project-specific parameters as described in Sect. 4.1. We are interested in the improvement of such a parameterized model compared to a model using the average commit behavior of developers. This helps us to find out how important the distribution of changes of developers is for the growth of the software as well as the structural evolution. We measure this effect by comparing the two size related metrics NOC and NOF as well as the two network metrics modularity and density over the time, i.e., for each year and

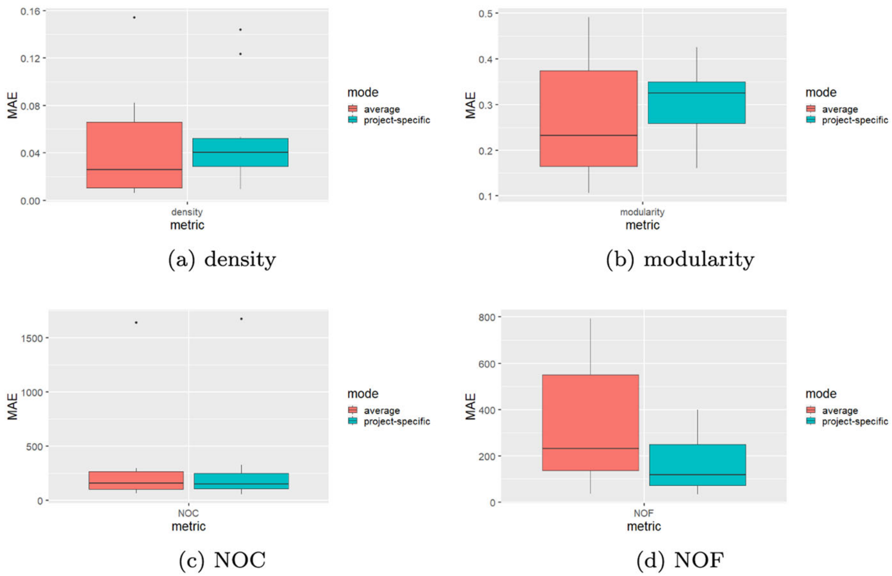


Fig. 15 Boxplots of MAEs for all projects

simulation type. Then, this two observation sequences are pairwise compared to the empirical observations. Therefore, we follow our evaluation scheme from Sect. 5.2. To determine the deviation from the simulation to the empirical values, we calculate the MAE and RMSE for each metric and project.

Figure 15 shows the distribution of errors (MAE) among projects for each metric. Here, it can already be seen that there generally is a larger margin for the average simulation than for the project-specific one. This is due to the fact that an average commit behavior may be appropriate for some projects, but may produce a larger error in other projects, e.g., with many low or high contributing developers. Overall, the differences between the two performances do not seem to be huge. We observed the same for the RMSE values (see "Appendix A". For measuring the significance, we proceed with step two from our evaluation scheme. The Shapiro-Wilk test resulted in not normally distributed for four out of sixteen populations. Thus, we check the difference between the population using the Wilcoxon Signed Rank test (step 3).

Table 2 lists all retrieved median values. For this comparison, we could not detect any significant results. Although no significant improvement can be determined, we observed that the simulation using the average change probabilities of developers performs best for mid-size projects. We illustrate this finding in Fig. 16. The NOF correspond to the number of nodes in the change coupling graph. The yearly evolution of this metric for the different simulation models as well as the empirical growth is shown. The figures support the assumption that for smaller projects (Gora) measured in the total number of commits, the average change behavior produces too few file actions of developers, because the developers contribution frequency is lower. As expected, mid-size projects like Zookeeper perform well with the average simulation

Table 2 Median values of MAE and RMSE for the average and the project-specific simulation model

Metric	MAE specific simulation	MAE average simulation	RMSE specific simulation	RMSE average simulation
Density	0.040	0.026	0.074	0.036
Modularity	0.325	0.232	0.335	0.24
NOC	149	160	173	178
NOF	118	231	140	268

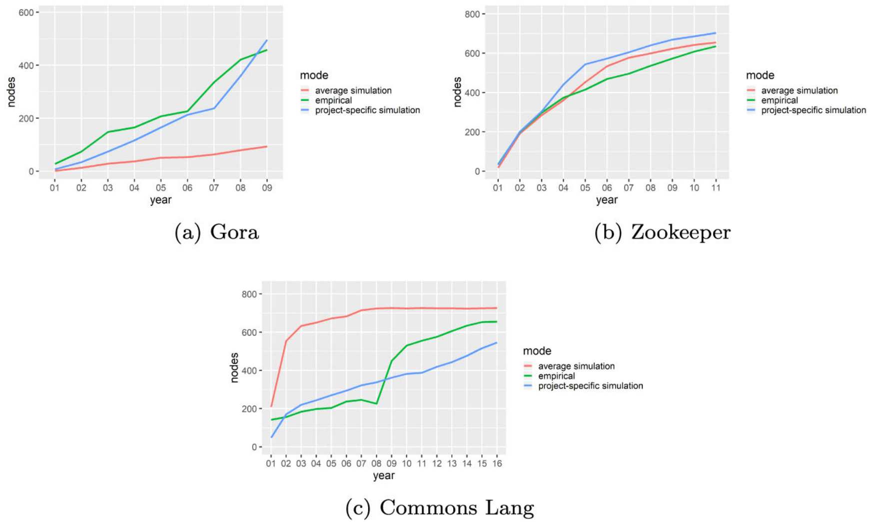


Fig. 16 NOF for a projects of different sizes

and for large projects (Commons Lang) it has the opposite effect. There, the average change probabilities result in too many file changes.

Thus, we can answer $RQ1$ the following:

> *In average, a project-specific initialized simulation model does not yield better results than a simulation model where the file change behavior is parameterized over all projects included in the case study. The average simulation model performs best for projects with a medium amount of effort spent by the developers.*

RQ2: How accurate is a long-term prediction of software evolution trends compared to a short-term prediction?

For $RQ2$, we proceed the same way as for $RQ1$. For this question, we compare the project-specific model with the model which takes a complete snapshot after one third of the current lifespan as input. We use the same ten open source software projects for this comparison and shorten the observation sequence produced by the project-specific model that it fits the snapshot observations. The comparison starts after one year since the snapshot initialization. Once again we follow our evaluation scheme described in Sect. 5.2. In doing so, we first calculate our error measures.

In Fig. 17, the distribution of the MAE for all metrics is shown. Thus, it appears that whereas the project growth metrics do not improve, the error rate for the network metrics is lower for the snapshot simulation. The Shapiro Wilk test identifies three out of sixteen populations as not normal. Hence, we apply the Wilcoxon test again.

Table 3 shows all median values. Significant differences in the populations are bold. Thereby, a significant improvement in the modularity of the change coupling network can be confirmed. This meets our expectations in so far that commit frequency

Table 3 Median values of MAE and RMSE for the project-specific simulation model and the snapshot simulation model

Metric	MAE specific simulation	MAE snapshot simulation	RMSE specific simulation	RMSE snapshot simulation
Density	0.0235	0.01	0.028	0.013
Modularity	**0.325**	**0.185**	**0.335**	**0.192**
NOC	149	180	173	204
NOF	118	116	140	133

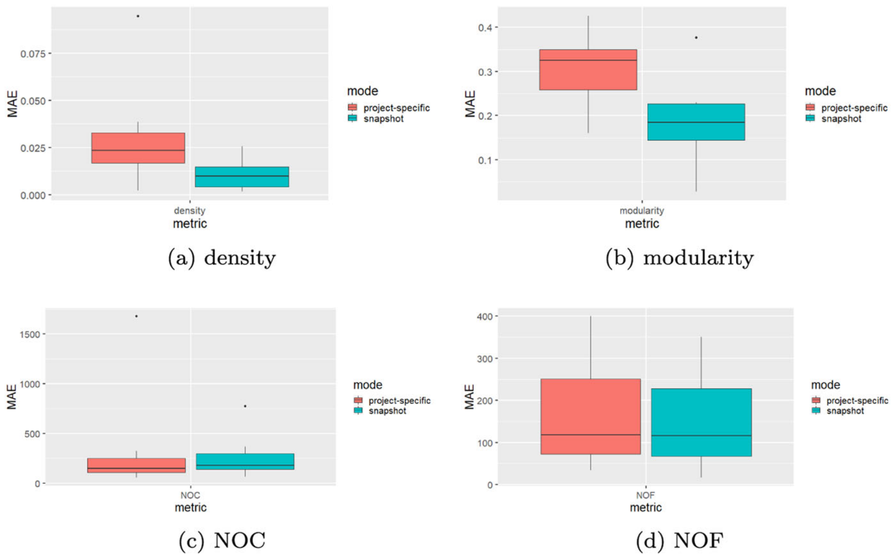


Fig. 17 Boxplots of MAEs for all projects

can be simulated easier than the distribution of changes among the files, since the intention of the developers as not directly visible from mining. As the density of the retrieved change coupling graphs of all OSS projects from the study is in almost all cases below 0.25 or as in one case slightly higher in the beginning of the project, we assume this to be the reason for no significance in the results in terms of the density. Moreover, the density of change coupling graphs evolves slowly without getting noticeably higher. From the perspective of software engineering, a steady and low density is good because it indicates a separation of structural connected entities in the commit behavior. Though, a high modularity can indicate hard to maintain areas.

We answer $RQ2$ the following:

In average, a simulation model based on a snapshot can yield better results than only a parameterization with project-specific parameters. Most striking, the graph metrics of the change coupling graph can be improved. The error rate for the modularity yields significantly better results for the snapshot based model.

5.4 Discussion

In this paper, we compared different simulation models for the prediction of future trends in software projects. We were especially interested in the growth of the system in the number of files and the number of commits as well as the structural evolution expressed by change coupling graphs which describe file dependencies by common changes. We suggested to parameterize software evolution models with project-specific parameters as well as to utilize a whole snapshot of a project as initialization. To achieve this, we proposed a mining framework closely working together

with a simulation framework. For evaluation, we tested different initializations of software project instances. We performed two experiments in our case study, one for each research question. For the first question, we compared the project-specific simulation with the simulation model using average change probabilities of developers. Here, we observed no significant differences considering the system growth in NOC and NOF as well as the change coupling evolution in density and modularity.

Hence, we reject **H1** that a project-wise simulation model can yield significant better results in terms of NOC and NOF than our baseline model.

This establishes a surprisingly good result for our baseline model, since it gives evidence that the general commit behavior and file growth of OSS systems can be mirrored by quite simple assumptions behind the agent-based simulation model.

The second research question compared the project-specific simulation with a simulation model which takes the whole state of a project snapshot after one third of the lifespan as input. Here, we were able to prove significance for the metric modularity for the snapshot initialization.

We reject **H2** that a snapshot-based simulation model can yield significant better results than the project-wise simulation model in terms of the modularity and the density of the change coupling graph. Considering only the modularity of the change coupling network, we can accept the hypothesis.

Implications for researchers and project managers

To summarize, these findings showed that for a prediction of the general trend the baseline as well as the project-wise simulation models perform well. For deeper investigations of software quality, e.g., to control change coupling since a high change coupling can indicate hard to maintain areas and structural issues such as architecture decay (D'Ambros et al. 2009), the usage of the snapshot-based model can be beneficial.

The overall goal of this study was to make the validation of simulation models using real data more feasible. We believe that we reach this goal by the automation of the parameter estimation steps and the usage of an analysis script. In doing so, we work on the empirical data and have the possibility to process the same desired aspects to predict in the simulation framework which already provides the foundation for the validation. Still, if, e.g., the research focus is changed, it may require an adaption of the analysis script and maybe also of the simulation itself. From our experience, we recommend using empirical data for the calibration and validation of simulation models.

Additionally, we think that the feedback loop established by testing different parameters, e.g., adding a developer to the project, can help project managers in decision making. Coming back to our motivating scenario, the project manager can use the predicted number of files and commits under the current team constellation as decision help for the onboarding process.

Compared to other prediction models which exist to predict, e.g., the effort of developers, we present a more global view on the software project involving side

effects of distinct behavior. Considering other simulation models giving a global view on software evolution, the models are too diverse to be compared directly.

5.5 Threats to validity

In this section, we address some threats to validity concerning our work. Considering the internal validity, we recognize that the evaluation depends on the choice of the metrics to describe the software evolution process. Thereby, it also depends on our conception of software evolution. For the design of a simulation model it is required to find a balance between model assumptions going into the model and knowledge that comes from outside. Thus, we had to identify the most important factors driving software evolution to prevent the model from becoming too complex. As such, developers and their behavior are very important to model accurately (Girba et al. 2005). We already argued why we use our developer classification in Sect. 4.2. We measure the growth of the system in NOF and NOC, because combined they reflect the trade-off between the output and the frequency of work of developers. Also, we use change coupling graphs and network metrics which are a common choice to model co-changes in literature. Still, other metrics could reflect other growth and activity trends of software evolution, e.g., size of the commits in LOC, number of active developers, number of open bugs, or the number of pull requests.

The choice of the projects affects the external validity. For our case study, we selected OSS projects of different size, duration, and team constellation to guarantee representativeness and respect diversity of projects. Moreover, we consider different growth trends via our project selection. Still, the findings from our case study may not be representative for other projects and their context, especially for closed source projects. The differences can lie in the motivation, background of developers, organization, timeline, or given process models. Hence, it would be worth investigating the validity of the results for other project types.

In this study, we were specially interested in the validity of simulation studies which raise special threats to validity in software engineering. De Franca et al. (2016) take possible threats in software engineering studies using simulation techniques into account and identified a set of guidelines to deal with them. For such studies, the validity testing has to take place on different levels. First, since the simulation naturally underlies some stochastic processes, it is needed to test the variance of the different simulation runs and beyond several parameter sets. The stochastic itself also has to be tested for the usage. The next level of validation specifies the balance between model assumptions and knowledge from outside the model. An imbalance can diminish the comprehensibility. By our mining process we satisfy two levels at once: First, empirical findings are validated in advance before becoming part of the model and second, we establish validity by the use of real software projects. However, the ground truth generated this way again relies on the chosen projects. Moreover, changes in the assumptions regarding the design of the simulation model for software evolution may also change the results although we tried to keep the rate of inherent knowledge low.

6 Conclusion and future work

This paper presents an approach to use and validate agent-based simulation models for the assessment and prediction of software evolution. Our investigation starts with the main driver of software evolution, the developers. For the parameterization of the simulation model, we identified developer roles, change probabilities and frequencies, related file growth, and file dependencies. The process of software evolution is controlled by the behavior of developers as active agents in the simulation. Starting from this general description of software evolution, we build four simulation models to evaluate how much project information is needed to avoid prediction errors. The novelty of this approach lies in the combination of different techniques to achieve this goal. For the parameter estimation, we use repository mining which enriches the simulation with knowledge retrieved from empirical data.

Since a simulation can only be as good as its model, it is required to create the model and choose its parameters carefully. Good models require some balance between empirical realism and simplicity (Maria 1997). A big part of our model depends on the work of the developers, since they create, update, and delete the files. This work controls the file evolution of the software under simulation and thus, the size of the change coupling graph. The structural evolution of this graph is based on assumptions of the work of developers, e.g., that they tend to work on certain areas of the software again if they already changed them before. As such, our model is fed with heuristics from real software projects, e.g., how often the distinct developer types core, major, and minor developers contribute to the project, but also some assumptions are made. Our case studies showed that such a simulation can produce realistic results and, thus, be very useful for trend prediction or to play the what-if game. In addition, we found that a model initialized with a snapshot after one third of the project can yield better prediction results, especially for the representation of the change coupling graph.

Possible directions of future work include a change of the focus of targeted software evolution scenarios which can have an impact on the results and the scope of the simulation. The current simulation model is extensible to other research questions, one direction could for example be the impact of team structure dynamics. Therefore, we plan to investigate the impact of phases with very low or high developer activity and identify possible causes, e.g., a rise of activity before a release is planned. For this, we aim to use the dynamic developer contribution model introduced in Honsel et al. (2016b).

Since the current model does not reflect bug introduction properly, it would be worth investigating how this could be modeled based on the changes by the developers. Therefore, we also want to map the changes performed in a commit to certain tasks and their difficulty. Then, based on the experience of the developer and the task she is working on a probability can be calculated of the bug introduction. But, for this, we will also need a task assignment strategy.

To maximize the generality of our results, we also plan to introduce more projects of different size and complexity. We would especially be interested in experiments in the closed source context, because we would expect some changes in the work of developers there. Vice versa, it would be worth investigating of the specialty of

projects, e.g., whether sudden periods of inactivity or major design changes could be predicted with a high certainty. [4]

Acknowledgements We would like to thank the Simulation Science Center Clausthal/Göttingen for partially funding our work as part of the project *Simulation-Based Quality Assurance for Software Systems*.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

A Boxplots for RMSE for *RQ1*

Fig. 18 provides details about RQ1. It shows the boxplots for RSME for all projects under the given setting.

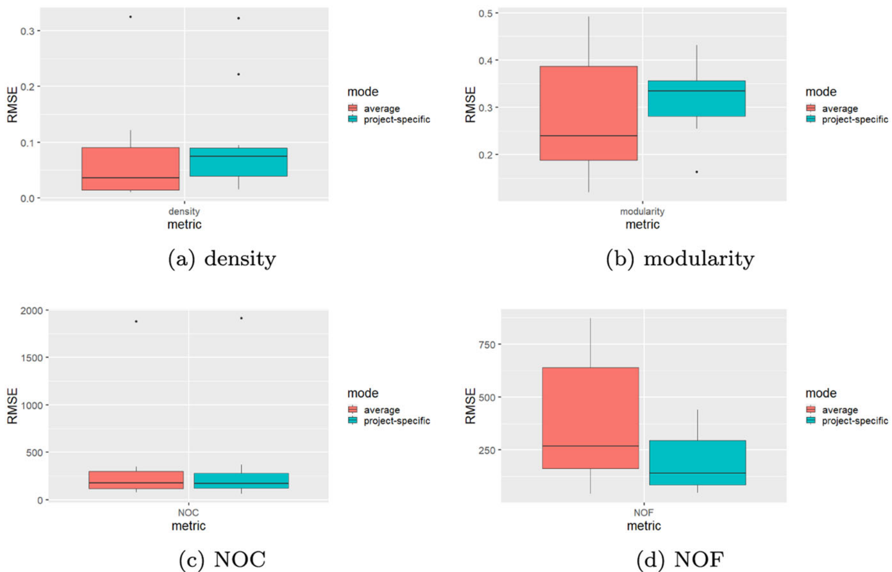
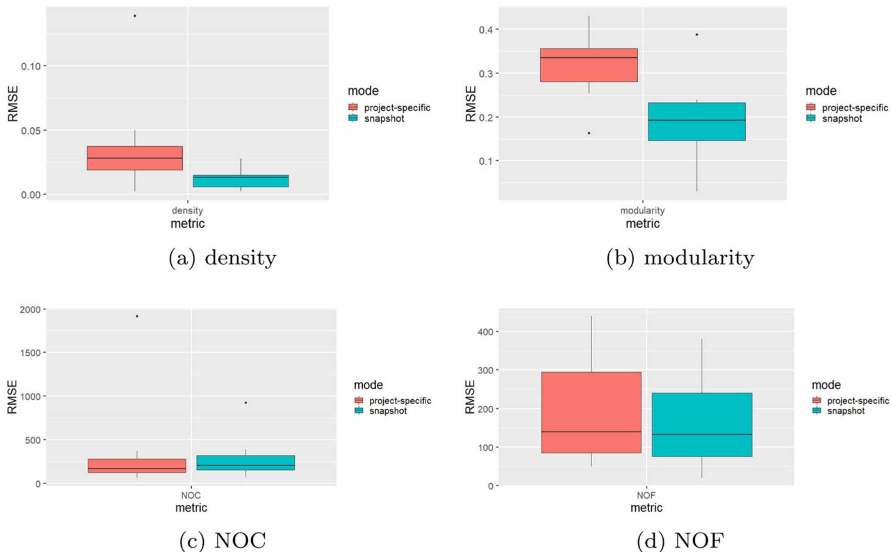


Fig. 18 Boxplots of RMSEs for all projects

[4] https://www.simzentrum.de/en/research-projects/.

B Boxplots for RMSE for *RQ2*

In Fig. 19 the boxplots for RSME for all projects containing details about RQ2 are shown.



(a) density (b) modularity (c) NOC (d) NOF

Fig. 19 Boxplots of RMSEs for all projects

References

Alfayez, R., Behnamghader, P., Srisopha, K., Boehm, B.: How does contributors involvement influence open source systems. In: 2017 IEEE 28th Annual Software Technology Conference (STC). https://doi.org/10.1109/STC.2017.8234462 (2017)

Ali, S.M., Doolan, M., Wernick, P., Wakelam, E.: Developing an agent-based simulation model of software evolution. Information and Software Technology. https://doi.org/10.1016/j.infsof.2017.11.013 (2018)

Amrit, C., van Hillegersberg, J.: Exploring the impact of socio-technical core-periphery structures in open source software development. J. Inf Technol. (2010). https://doi.org/10.1057/jit.2010.7

Ball, T., Kim, J.M., Porter, A.A., Siy, H.P.: If your version control system could talk. In: ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering (1997)

Bastian, M., Heymann, S., Jacomy, M., et al.: Gephi: an open source software for exploring and manipulating networks. In: Proc. of the 3rd Intern. AAAI Conf. on Weblogs and Social Media (ICWSM) (2009)

Ben, X., Beijun, S., Weicheng, Y.: Mining developer contribution in open source software using visualization techniques. In: Proceedings of the Third International Conference on Intelligent System Design and Engineering Applications (ISDEA) (2013). https://doi.org/10.1109/ISDEA.2012.223

Bhattacharya, P., Iliofotou, M., Neamtiu, I., Faloutsos, M.: Graph-based analysis and prediction for software evolution. In: Proceedings of the 34th Intern. Conf. on Softw. Eng. (ICSE). IEEE (2012)

Bhattacharya, P., Neamtiu, I., Faloutsos, M.: Determining developers' expertise and role: a graph hierarchy-based approach. In: ICSME, IEEE Computer Society, pp 11–20 (2014)

Bird, C., Gourley, A., Devanbu, P., Gertz, M., Swaminathan, A.: Mining email social networks. In: Proceedings of the 2006 International Workshop on Mining Software Repositories, ACM, New York, NY, USA, MSR '06, pp. 137–143 (2006). https://doi.org/10.1145/1137983.1138016

Bird, C., Nagappan, N., Murphy, B., Gall, H., Devanbu, P.: Don't touch my code!: Examining the effects of ownership on software quality. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ACM, ESEC/FSE '11, pp. 4–14 (2011). https://doi.org/10.1145/2025113.2025119

Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **2008**(10), P10008 (2008). http://stacks.iop.org/1742-5468/2008/i=10/a=P10008

Caglayan, B., Bener, A.B., Miranskyy, A.: Emergence of developer teams in the collaboration network. In: 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE). https://doi.org/10.1109/CHASE.2013.6614729 (2013)

Crowston, K., Howison, J.: Hierarchy and centralization in free and open source software team communications. Knowl. Technol. Policy **18**(4), 65–85 (2006)

D'Ambros, M., Lanza, M., Robbes, R.: On the relationship between change coupling and software defects. In: Proc. of the 16th Working Conf. on Rev. Eng., IEEE Computer Society (2009)

Fernandez-Ramil, J., Lozano, A., Wermelinger, M., Capiluppi, A.: Empirical studies of open source evolution. In: Mens, T., Demeyer, S. (eds.) Software Evolution: State-of-the-Art and Research Advances. Springer Verlag (2008)

Fortunato, S.: Community detection in graphs. Physics Reports 486. https://doi.org/10.1016/j.physrep.2009.11.002 (2010)

de França, B.B.N., Travassos, G.H.: Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. Empirical Software Engineering (2016). https://doi.org/10.1007/s10664-015-9386-4

Gall, H., Hajek, K., Jazayeri, M.: Detection of logical coupling based on product release history. In: Proc. of the Intern. Conf. on Softw. Maint. (ICSM), IEEE Computer Society (1998)

García-García, J., Enríquez, J., Ruiz, M., Arívalo, C., Jiménez-Ramérez, A.: Software process simulation modeling: systematic literature review. Computer Standards & Interfaces (2020). https://doi.org/10.1016/j.csi.2020.103425

Girba, T., Kuhn, A., Seeberger, M., Ducasse, S.: How developers drive software evolution. In: Proceedings of the Eighth International Workshop on Principles of Software Evolution (2005)

Godfrey, M.W., Tu, Q.: Evolution in open source software: a case study. In: Proc. Int'l Conf. Software Maintenance (ICSM) (2000)

Goeminne, M., Mens, T.: A comparison of identity merge algorithms for software repositories. Science of Computer Programming (2013). https://doi.org/10.1016/j.scico.2011.11.004

Gousios, G., Kalliamvakou, E., Spinellis, D.: Measuring developer contribution from software repository data. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories. https://doi.org/10.1145/1370750.1370781 (2008)

Herbold, S., Trautsch, A., Trautsch, F.: Issues with szz: an empirical assessment of the state of practice of defect prediction data collection. arXiv preprint arXiv:191108938 (2019) http://arxiv.org/abs/1911.08938v1

Herbold, V.: Mining developer dynamics for agent-based simulation of software evolution. Ph.D. thesis, Georg-August-Universität Göttingen. http://hdl.handle.net/21.11130/00-1735-0000-0003-C15C-C (2019)

Herbold, V.: Asej—replication kit. online. https://github.com/vhonsel/sim_data_ASEJ_2020 (2020)

Herraiz, I., Robles, G., Gonzalez-Barahon, J.u.M.: Comparison between slocs and number of files as size metrics for software evolution analysis. In: Proceedings of the Conference on Software Maintenance and Reengineering, IEEE Computer Society, CSMR '06. http://dl.acm.org/citation.cfm?id=1116163.1116405 (2006)

Herzig, K., Zeller, A.: The impact of tangled code changes. In: Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, MSR '13. http://dl.acm.org/citation.cfm?id=2487085.2487113 (2013)

Hindle, A., German, D.M., Godfrey, M.W., Holt, R.C.: Automatic classication of large changes into maintenance categories. In: 2009 IEEE 17th International Conference on Program Comprehension. https://doi.org/10.1109/ICPC.2009.5090025 (2009)

Honsel, D.: Development of agent-based simulation models for software evolution. PhD thesis, Georg-August-Universität Göttingen. http://hdl.handle.net/21.11130/00-1735-0000-0005-1318-B (2019)
Honsel, D.: Simparameter – estimation of simulation parameters. online https://github.com/dhonsel/SimParameter (2020a)
Honsel, D.: Simse – simulation of software evolution. online https://github.com/dhonsel/SimSE (2020b)
Honsel, D., Herbold, V., Welter, M., Grabowski, J., Waack, S.: Monitoring software quality by means of simulation methods. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, ESEM '16. https://doi.org/10.1145/2961111.2962617 (2016a)
Honsel, V., Honsel, D., Grabowski, J.: Software process simulation based on mining software repositories. The Third International Workshop on Software Mining (2014)
Honsel, V., Honsel, D., Herbold, S., Grabowski, J., Waack, S.: Mining software dependency networks for agent-based simulation of software evolution. The Fourth International Workshop on Software Mining (2015)
Honsel, V., Herbold, S., Grabowski, J.: Hidden markov models for the prediction of developer involvement dynamics and workload. In: 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE) (2016b)
Huang, S.K., Liu, K.m.: Mining version histories to verify the learning process of legitimate peripheral participants. SIGSOFT Softw Eng Notes. https://doi.org/10.1145/1082983.1083158 (2005)
Joblin, M., Apel, S., Hunsen, C., Mauerer, W.: Classifying developers into core and peripheral: An empirical study on count and network metrics. In: Proceedings of the 39th International Conference on Software Engineering, IEEE Press, ICSE '17. https://doi.org/10.1109/ICSE.2017.23 (2017)
Khondhu, J., Capiluppi, A., Stol, K.J.: Is it all lost? a study of inactive open source projects. In: Open Source Software: Quality Verification. Springer Berlin Heidelberg (2013)
Kim, S., Whitehead, E.J., Zhang, Y.: Classifying software changes: clean or buggy? Software engineering. IEEE Transactions on. https://doi.org/10.1109/TSE.2007.70773 (2008)
Kocaguneli, E., Misirli, A.T., Caglayan, B., Bener, A.B.: Experiences on developer participation and effort estimation. In: EUROMICRO-SEAA. IEEE (2011)
Konopka, M., Navrat, P.: Untangling development tasks with software developer's activity. In: 2015 IEEE/ACM 2nd International Workshop on Context for Software Development. https://doi.org/10.1109/CSD.2015.10 (2015)
Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B.: Predicting the severity of a reported bug. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). https://doi.org/10.1109/MSR.2010.5463284 (2010)
Lehman, M.M.: Programs, life cycles, and laws of software evolution. Proc. IEEE **68**(9) (1980)
Li, Y., Tan, C.H., Teo, H.H.: Leadership characteristics and developers' motivation in open source software development. Inf. Manag. (2012)
Lima, J., Treude, C., Filho, F.F., Kulesza, U.: Assessing developer contribution with repository mining-based metrics. In: Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on. https://doi.org/10.1109/ICSM.2015.7332509 (2015)
Macal, C.M., North, M.J.: Tutorial on agent-based modeling and simulation. In: Proceedings of the 37th Conference on Winter Simulation, Winter Simulation Conference, WSC '05 (2005)
Macal, C.M., North, M.J.: Tutorial on agent-based modeling and simulation part 2: How to model with agents. In: Proceedings of the 38th Conference on Winter Simulation, Winter Simulation Conference, WSC '06 (2006)
Maria, A.: Introduction to modeling and simulation. In: Proceedings of the 29th conference on Winter simulation. IEEE Computer Society (1997)
Meneely, A., Williams, L., Snipes, W., Osborne, J.: Predicting failures with developer networks and social network analysis. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, SIGSOFT '08/FSE-16. https://doi.org/10.1145/1453101.1453106 (2008)
Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of open source software development: Apache and mozilla. ACM Trans. Softw. Eng. Methodol. **11**(3), 309–346 (2002). https://doi.org/10.1145/567793.567795
North, M.J., Collier, N.T., Ozik, J., Tatara, E.R., Macal, C.M., Bragen, M., Sydelko, P.: Complex adaptive systems modeling with repast simphony. Complex Adaptive Systems Modeling (2013)

Paulson, J.W., Succi, G., Eberlein, A.: An empirical study of open-source and closed-source software products. IEEE Trans. Softw. Eng. (2004). https://doi.org/10.1109/TSE.2004.1274044
Rahman, F., Devanbu, P.: Ownership, experience and defects: a fine-grained study of authorship. In: Proc. of the 33rd Intern. Conf. on Softw. Eng. (ICSE). ACM (2011)
Robles, G., Amor, J.J., Gonzalez-Barahona, J.M., Herraiz, I.: Evolution and growth in large libre software projects. In: Eighth International Workshop on Principles of Software Evolution (IWPSE'05). IEEE (2005)
Sargent, R.G.: Verification and validation of simulation models. In: Proceedings of the Winter Simulation Conference, Winter Simulation Conference, WSC '11. http://dl.acm.org/citation.cfm?id=2431518.2431538 (2011)
Shaffer, J.P.: Multiple hypothesis testing. Annu. Rev. Psychol. (1995)
Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika (1965). https://doi.org/10.1093/biomet/52.3-4.591
Shihab, E., Hassan, A.E., Adams, B., Jiang, Z.M.: An industrial study on the risk of software changes. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. https://doi.org/10.1145/2393596.2393670 (2012)
Smith, N., Capiluppi, A., Fernández-Ramil, J.: Agent-based simulation of open source evolution. In: Software Process Improvement and Practice (2006)
Terceiro, A., Rios, L.R., Chavez, C.: An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. In: Software Engineering (SBES), 2010 Brazilian Symposium on. IEEE (2010)
Trautsch, F., Herbold, S., Makedonski, P., Grabowski, J.: Addressing problems with replicability and validity of repository mining studies through a smart data platform. Empir. Softw. Eng. (2018). https://doi.org/10.1007/s10664-017-9537-x
Turski, W.M.: Reference model for smooth growth of software systems. IEEE Trans. Softw. Eng. (1996) http://dl.acm.org/citation.cfm?id=235681.235686
Wiese, I.S., Kuroda, R.T., Re, R., Oliva, G.A., Gerosa, M.A.: An empirical study of the relation between strong change coupling and defects using history and social metrics in the apache aries project. In: Open Source Systems: Adoption and Impact, Springer International Publishing (2015)
Wilcoxon, F.: Individual comparisons by ranking methods. Biom. Bull. (1945) http://www.jstor.org/stable/3001968
Willmott, C.J.: Some comments on the evaluation of model performance. Bull. Am. Meteorol. Soc. (1982). https://doi.org/10.1175/1520-0477(1982)063%3c1309:SCOTEO%3e2.0.CO;2
Yamauchi, K., Aman, H., Amasaki, S., Yokogawa, T., Kawahara, M.: An entropy-based metric of developer contribution in open source development and its application to fault-prone program analysis*. Int. J. Network. Distrib. Comput. (2018). https://doi.org/10.2991/ijndc.2018.6.3.1
Yu, L., Ramaswamy, S.: Mining cvs repositories to understand open-source project developer roles. In: Proceedings of the Fourth International Workshop on Mining Software Repositories, IEEE Computer Society, Washington, DC, USA, MSR '07. https://doi.org/10.1109/MSR.2007.19 (2007)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.