Monitoring Software Quality by Means of Simulation Methods

Daniel Honsel, Verena Honsel, Marlon Welter, Stephan Waack, and Jens Grabowski

Institute of Computer Science
Georg-August-Universität Göttingen

Ciudad Real, September 8th, 2016
Outline

1 Introduction

2 Agent-Based Simulation Model

3 Case Study and Results

4 Conclusion
1. Introduction
Motivation

**Goal:** Evaluating the quality of a project under simulation according to a specific question.
Approach: interaction of 3 research areas

Starting with a concrete question: **What happens if the core developer leaves the project?**

**Mining software repositories:** Estimate model parameters  
*Source:* open source repositories

**Agent-based modeling and simulation:** Create a model that answers the question  
- Adjust it with mined parameters  
- Running the simulation generates, for example, software dependency graphs

**Automated Assessment:** Evaluate the simulation results with Conditional Random Fields
Approach: interaction of 3 research areas

Starting with a concrete question: What happens if the core developer leaves the project?

Mining software repositories: Estimate model parameters

Source: open source repositories

Agent-based modeling and simulation: Create a model that answers the question

■ Adjust it with mined parameters
■ Running the simulation generates, for example, software dependency graphs

Automated Assessment: Evaluate the simulation results with Conditional Random Fields
Introduction

Approach: interaction of 3 research areas

Starting with a concrete question: **What happens if the core developer leaves the project?**

**Mining software repositories:** Estimate model parameters

*Source:* open source repositories

**Agent-based modeling and simulation:** Create a model that answers the question

- Adjust it with mined parameters
- Running the simulation generates, for example, software dependency graphs

**Automated Assessment:** Evaluate the simulation results with Conditional Random Fields

Daniel Honsel, Georg-August-Universität Göttingen
Approach: interaction of 3 research areas

Starting with a concrete question: What happens if the core developer leaves the project?

Mining software repositories: Estimate model parameters
   Source: open source repositories

Agent-based modeling and simulation: Create a model that answers the question
   ■ Adjust it with mined parameters
   ■ Running the simulation generates, for example, software dependency graphs

Automated Assessment: Evaluate the simulation results with Conditional Random Fields
2. Agent-Based Simulation Model
Which entities do we have to consider?

- **Software entities** are passive agents
- **Bugs** are passive agents
- **Developers** are active agents, their commits are responsible for the evolutionary process.
  - Update
  - Create
  - Delete
  - Bugfix
Agent-based model for software evolution

Environment
- fileCount : Integer

Bug
- dateOfCreation : Real
- computeLifespan() : Real

Category
- MajorBug
- NormalBug
- MinorBug

SoftwareEntity
- owner : Developer
- numberOfChanges : Integer
- numberOfAuthors : Integer
- couplingDegree : Integer
- computeLabelValue() : Real

Developer
- numberOfCommits : Integer
- numberOfFixes : Integer
- createFiles()
- updateFiles()
- deleteFiles()
- bugFix()

Maintainer
CoreDeveloper
MajorDeveloper
MinorDeveloper

works on
Which dependencies are considered?

Dependencies are represented as networks. The following ones are modeled:

**Developer – Entity:** A weighted graph representing the work of developers on entities. Provides owner, number of authors and changes.

**Bug – Entity:** Bug assignment to an entity.

**ChangeCoupling:** Dependencies between entities that are changed together several times. Input for the automated assessment.
Which dependencies are considered?

Dependencies are represented as networks. The following ones are modeled:

**Developer – Entity:** A weighted graph representing the work of developers on entities. Provides owner, number of authors and changes.

**Bug – Entity:** Bug assignment to an entity.

**ChangeCoupling:** Dependencies between entities that are changed together several times. Input for the automated assessment.
Which dependencies are considered?

Dependencies are represented as networks. The following ones are modeled:

**Developer – Entity:** A weighted graph representing the work of developers on entities. Provides owner, number of authors and changes.

**Bug – Entity:** Bug assignment to an entity.

**ChangeCoupling:** Dependencies between entities that are changed together several times. Input for the automated assessment.
Which dependencies are considered?

Dependencies are represented as networks. The following ones are modeled:

**Developer – Entity:** A weighted graph representing the work of developers on entities. Provides owner, number of authors and changes.

**Bug – Entity:** Bug assignment to an entity.

**ChangeCoupling:** Dependencies between entities that are changed together several times. Input for the automated assessment.
Commit behavior

- **Number** of software entities to be created, updated, or deleted have to be determined.

- **Entity selection** for updates:
  - Select *first entity* randomly.
  - *Further entities* will be selected *based on* information about the *first* one and dependency networks.

- **Apply the commit**:
  - *Change involved entities and networks*.
  - *Bugfix* if possible.
Commit behavior

- **Number** of software entities to be created, updated, or deleted have to be determined

- Entity **selection** for updates
  - Select **first** entity randomly
  - Further entities will be selected **based on** information about the **first** one and dependency networks

- Apply the **commit**
  - Change involved **entities** and **networks**
  - **Bugfix** if possible
Commit behavior

- **Number** of software entities to be created, updated, or deleted have to be determined.

- **Entity selection** for updates
  - Select *first* entity randomly
  - *Further entities* will be selected *based on* information about the *first* one and dependency networks

- **Apply the commit**
  - Change involved *entities and networks*
  - Bugfix *if possible*
Commit behavior

- **Number** of software entities to be created, updated, or deleted have to be determined

- **Entity selection** for updates
  - Select *first* entity randomly
  - *Further entities* will be selected *based on* information about the *first* one and dependency networks

- **Apply the commit**
  - *Change* involved *entities* and *networks*
  - *Bugfix* if possible
What means quality for the assessment?

Entity-wise **label** quantifies bugs assigned to a software entity.

- **Label-value** is **product** of bug factors assigned to a entity
  - Label-value initialized with 1
  - Each bug type has a factor < 1

- Each entity is preliminary labeled as **acceptable** (Label-value > 0.8) or **problematic** (otherwise).

- Simulated ChangeCoupling graph containing the labels serves as input for CRF assessment.
What means quality for the assessment?

Entity-wise label quantifies bugs assigned to a software entity.

- **Label-value** is **product** of bug factors assigned to a entity
  - Label-value initialized with 1
  - Each bug type has a factor $< 1$

- Each entity is preliminary labeled as **acceptable** (Label-value $> 0.8$) or **problematic** (otherwise).

- Simulated ChangeCoupling graph containing the labels serves as input for CRF assessment.
What means quality for the assessment?

Entity-wise label quantifies bugs assigned to a software entity.

- **Label-value** is product of bug factors assigned to a entity
  - Label-value initialized with 1
  - Each bug type has a factor < 1

- Each entity is preliminary labeled as **acceptable** (Label-value > 0.8) or **problematic** (otherwise).

- Simulated ChangeCoupling graph containing the labels serves as input for CRF assessment.
What means quality for the assessment?

Entity-wise **label** quantifies bugs assigned to a software entity.

- **Label-value** is **product** of bug factors assigned to a entity
  - Label-value initialized with 1
  - Each bug type has a factor < 1

- Each entity is preliminary labeled as **acceptable** (Label-value > 0.8) or **problematic** (otherwise).

- Simulated ChangeCoupling graph containing the labels serves as input for CRF assessment.
3. Case Study and Results
3 Case Study and Results

Simulate the effect of a lost core developer

We have
- Simulation model parametrized according to K3b

Figure: Assessment of K3b (CRF category labeling, green nodes are acceptable and red ones are problematic).
Quality trend of other projects

Change simulation parameters only for effort, size, and duration of other projects.

- Checked repeatable results successfully for Log4J and Kate
- Trend of the two simulation runs is similar as for K3b.
- With lost core developer 12% − 20% fewer bugs are fixed.
4. Conclusion
Conclusion

We are able to assess the quality of software under simulation

- Parameters to adjust: effort, size, and duration.

Next steps to improve the simulation model:

- Bug introducing related to commits
  - Depending on the ownership and coupling degree
- Reduce randomness when selecting entities for a commit
  - BDI: developers formulate goals and build plans to reach them
  - Goals: add feature, bugfix, reduce complexity, …
Conclusion

We are able to assess the quality of software under simulation

- Parameters to adjust: effort, size, and duration.

Next steps to improve the simulation model:

- Bug introducing related to commits
  - Depending on the ownership and coupling degree

- Reduce randomness when selecting entities for a commit
  - BDI: developers formulate goals and build plans to reach them
  - Goals: add feature, bugfix, reduce complexity, ...
Conclusion

We are able to assess the quality of software under simulation

- Parameters to adjust: effort, size, and duration.

Next steps to improve the simulation model:

- Bug introducing related to commits
  - Depending on the ownership and coupling degree

- Reduce randomness when selecting entities for a commit
  - BDI: developers formulate goals and build plans to reach them
  - Goals: add feature, bugfix, reduce complexity, …


Literature

V. Honsel.
Statistical learning and software mining for agent based simulation of software evolution.
In Doctoral Symposium at the 37th International Conference on Software Engineering (ICSE), 2015.

Verena Honsel, Steffen Herbold, and Jens Grabowski.
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), 2016.

Verena Honsel, Daniel Honsel, and Jens Grabowski.
Software process simulation based on mining software repositories.
In ICDM Workshop, 2014.

Verena Honsel, Daniel Honsel, Jens Grabowski, and Stephan Waack.
Developer Oriented and Quality Assurance Based Simulation of Software Processes.

Verena Honsel, Daniel Honsel, Steffen Herbold, Jens Grabowski, and Stephan Waack.
Mining software dependency networks for agent-based simulation of software evolution.

Gerhard Weiss.
Multiagent Systems.