Software Evolution – Reader

Edition 2020/2021 – Version 0.25

Paul Klint [1 4], Jurgen Vinju [1 4 7], Tijs van der Storm (guest lecturer) [1 6], Magiel Bruntink (guest lecturer) [3], Davy Landman (guest lecturer) [4], Vadim Zaytsev (guest lecturer) [5], Ana Oprescu (lab teacher) [2], Simon Baars (lab teacher) [2], and Riemer van Rozen (course coordinator, lecturer, lab teacher) [1 2]

[1]Software Analysis & Transformation, Centrum Wiskunde & Informatica
[2]Master of Software Engineering, University of Amsterdam
[3]Software Improvement Group
[4]SWAT.engineering
[5]University of Twente
[6]University of Groningen
[7]Eindhoven University of Technology

December 10, 2020

Abstract

This is a reader to the course Software Evolution. It describes course goals, a week-by-week course schedule, obligatory assignments and grading. In a nutshell, this manual explains how to pass this course[1]. Updates are provided on Canvas.

1 Course Overview

Software Evolution is a course in the Master of Software Engineering at the University of Amsterdam of 6 ECTS. We provide descriptions of course material in Section 1.2, required course activities in Section 1.3, a detailed schedule in Section 1.5, reading in Section 1.6 and the evaluation in Section 1.4. Section 2 describes the practical assignments. This document is continuously being updated: Please check Section 3 for a log of modifications over document versions. Please read this document carefully!

[1]This reader does not explain how to get the most out of this course, that's up to you.

Software Evolution's Evolution. The Software Evolution course itself has evolved over the years. Thanks and kudos for developing and maintaining this course and its assignments go to: Prof. Dr. Paul Klint, Prof. Dr. Jurgen Vinju, Dr. Magiel Bruntink and Dr. Vadim Zaytsev. Editor of the reader: Dr. ir. Riemer van Rozen.

1.1 Goals

The goals of the course are described in the study guide[2].

Exit qualification: *"The graduate masters the methods and techniques needed to analyze an existing software system and to enable it to evolve given changing requirements."*

Our objectives are three-fold:

- The first objective is to acquire an understanding and appreciation of the challenges posed by software maintenance and software evolution.
- The second objective is to learn about quality of software and source code and how it affects software maintenance and evolution.
- The final objective is to be able to select and also construct software analysis and software transformation tools to help obtain insight in software quality and to help improve software quality.

The course ties in closely with paper writing sessions where the objectives are to learn from academic literature, to develop curiosity, and to improve argumentation and writing skills.

1.2 Course Material

Slides & Papers. We provide a selection of scientific papers and lecture slides, which are available on Canvas. Additional papers can be found at the

- ACM Digital Library `http://www.acm.org/dl`
- IEEE Digital Library `http://ieeexplore.ieee.org` and `https://www.computer.org`

Rascal. For the practical lab assignments use metaprogramming language and language workbench Rascal[3]. Rascal has a built-in *Tutor* that provides explanations on concepts and interactive exercises for learning to apply language features. Two non-interactive versions are available online, an old version[4] and a new one[5]. Additionally, questions can be posed on Stackoverflow[6] using the `rascal` tag, and issues can be reported on GitHub[7].

[2] `https://studiegids.uva.nl/xmlpages/page/2020-2021/zoek-vak/vak/79025`
[3] `http://www.rascal-mpl.org`
[4] `http://tutor.rascal-mpl.org` (old tutor – may be outdated with recent unstable builds)
[5] `http://docs.rascal-mpl.org/unstable/TutorHome` (new tutor – not yet finished)
[6] `http://stackoverflow.com/questions/tagged/rascal`
[7] `https://github.com/usethesource/rascal/issues`

1.3 Required Course Activities

The course consists of activities related to reading scientific papers, discussing those papers, and attending lectures and the practical lab[8].

- **Reading: Study.** Students study a selection of scientific papers each week, as well as the slides that accompany the lecture. Please check the course schedule for detailed information in Section 1.5 and weekly reading in Section 1.6.
- **Paper Sessions: Read and write.** During the paper sessions students discuss papers with the lecturer. Special attention is given to analyzing scientific papers and writing reviews. As part of this course students are asked to write an annotated bibliography.
- **Lecture Part 1 – Seminar: Introduce and discuss a topic.** During the first part of the lecture on day 1 the lecturer introduces a subject by presenting a deck of slides. The group questions the concepts, problems and solutions and discusses them.
- **Lecture Part 2 – Practical Lab: Work on assignments.** During the second part of the lecture on day 1 and on day 2 students work on practical assignments. Students work on practical assignments in teams of two students. Every practical session students are encouraged to ask for feedback with the lecturer, improving their work over several iterations before finally handing it in before the deadline, and the assignments are graded.

1.3.1 Grading

The course grade is the average of three grades[9], for practical lab *Series 1* and *Series 2* and an individual grade for an *Annotated Bibliography* of papers you have studied[10]. Grades are calculated as follows.

```
grade(series1, annotated_bibliography, series2) =
    (grade(series1) + grade(annotated_bibliography) + grade(series2))/3
```

1.4 Evaluation

Just like the programming students produce during the practical labs, this course is also the product of iterative improvement. We request student feedback in two ways:

- suggestions during the course, for quick on-the-fly improvements to current and upcoming lectures.
- assessment at the end of the course, for improving the course structure, content and goals for the next year.

[8] **Note:** Lecture attendance is mandatory.
[9] **Note:** UvA's rounding rules apply.
[10] **Note:** Practical lab Series 0 is mandatory but not graded.

1.5 Course Schedule

Table 1 shows a week-by-week schedule of topics, lecture dates and lecturers. The columns *L* and *W* indicate the lecture week number and the calendar week number. The columns *Subject*, *Date* and *Lecturer* receptively show a brief description of the subjects for that week, the date of the lecture, and the name of the (guest) lecturer.

Table 2 shows reading, assignments and deadlines. The column *Reading* specifies which papers to study. The columns *Practical Lab* and *Deadlines* show which practical lab to work on during that week, and which assignments are due[11].

L	W	Subject	Date	Lecturer
1	44	Introduction to Software Evolution	Oct 26th	Riemer van Rozen
2	45	Meta-programming and Rascal	Nov 2nd	Tijs van der Storm
3	46	Software Metrics, the SIG Perspective	Nov 9th	Magiel Bruntink
4	47	CC & SLOC, SWAT.engineering	Nov 16th	Davy Landman
5	48	Clone Detection and Management	Nov 23rd	Riemer van Rozen
6	49	Legacy Software and Renovation	Nov 30th	Vadim Zaytsev
7	50	Model Evolution	Dec 7th	Riemer van Rozen

Table 1: Course Plan: Lecture Topics, Lecture Dates and Lecturers

L	W	Reading	Practical Lab	Deadlines
1	44	Mens [21]. Herraiz *et al.* [11].	Series 0 & 1	Series 0
2	45	Klint *et al.* [14].	Series 1	
3	46	Fenton [6]. Heitlager *et al.* [9]. Basili *et al.* [4].	Series 1	
4	47	Landman *et al.* [19]	Series 2	Series 1
5	48	Koschke [17]. Kapser and Godfrey [13].	Series 2	
6	49	Zaytsev [31].	Series 2	Series 2
7	50	Hermannsdörfer and Wachsmuth [10]. Alanen and Porres [1]. van Rozen and van der Storm [27].	Series 2	
8	51	no additional reading		Annotated bibliography

Table 2: Course Plan: Reading, Assignments and Deadlines

[11] **Note:** Please find the exact deadlines in the assignment descriptions

1.6 Reading

This is the list of papers students read during the course, organised week-by-week the papers are covered in the lectures. Each paper on this list from weeks 1–5 is to be included in the annotated bibliography that students create during the course. The reading from weeks 6 and 7 is recommended reading only.

Lecture 1

- T. Mens. "Software Evolution". In: ed. by T. Mens and S. Demeyer. Springer, 2008. Chap. 1. Introduction and Roadmap: History and Challenges of Software Evolution, pp. 2–11. ISBN: 978-3-540-76440-3. DOI: 10.1007/978-3-540-76440-3
- I. Herraiz et al. "The Evolution of the Laws of Software Evolution: A Discussion Based on a Systematic Literature Review". In: *ACM Comput. Surv.* 46.2 (Dec. 2013), pp. 1–28. ISSN: 0360-0300. DOI: 10.1145/2543581.2543595

Lecture 2

- P. Klint, T. v. d. Storm, and J. Vinju. "RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation". In: *Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2009, Edmonton, AB, Canada, September 20–21, 2009*. IEEE, 2009, pp. 168–177. ISBN: 978-0-7695-3793-1. DOI: 10.1109/SCAM.2009.28

Lecture 3

- N. Fenton. "Software Measurement: A Necessary Scientific Basis". In: *IEEE Transactions on Software Engineering* 20.3 (Mar. 1994), pp. 199–206. ISSN: 0098-5589. DOI: 10.1109/32.268921
- I. Heitlager, T. Kuipers, and J. Visser. "A Practical Model for Measuring Maintainability". In: *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*. 2007, pp. 30–39. DOI: 10.1109/QUATIC.2007.8
- V. and Basili, G. Caldiera, and H. D. Rombach. "Encyclopedia of Software Engineering". In: Wiley, 2002. Chap. Goal Question Metric (GQM) Approach. ISBN: 9780471028956. DOI: 10.1002/0471028959.sof142

Lecture 4

- D. Landman, A. Serebrenik, and J. J. Vinju. "Empirical Analysis of the Relationship between CC and SLOC in a Large Corpus of Java Methods". In: *30th IEEE International Conference on Software Maintenance and Evolution, ICSME 2014, Victoria, BC, Canada, September 29–October 3, 2014*. 2014, pp. 221–230. ISBN: 978-1-4799-6146-7. DOI: 10.1109/ICSME.2014.44[12]

[12] An extended journal version exists [18].

The original reading of this week has become optional, and is not required for the annotated bibliography:

- S. Erdweg et al. "The State of the Art in Language Workbenches: Conclusions from the Language Workbench Challenge". In: *Software Language Engineering – Proceedings of the 6th International Conference, SLE 2013, Indianapolis, IN, USA, October 26–28, 2013*. Ed. by M. Erwig, R. F. Paige, and E. Van Wyk. Vol. 8225. LNCS. Springer, 2013, pp. 197–217. ISBN: 978-3-319-02654-1. DOI: 10.1007/978-3-319-02654-1_11

Lecture 5

- R. Koschke. "Software Evolution". In: ed. by T. Mens and S. Demeyer. Springer, 2008. Chap. 2. Identifying and Removing Software Clones, pp. 15–36. ISBN: 978-3-540-76440-3. DOI: 10.1007/978-3-540-76440-3
- C. Kapser and M. W. Godfrey. ""Cloning Considered Harmful" Considered Harmful". In: *2006 13th Working Conference on Reverse Engineering*. Oct. 2006, pp. 19–28. DOI: 10.1109/WCRE.2006.1

Lecture 6 The following papers is recommended reading only, and are not required for the annotated bibliography assignment.

- V. Zaytsev. "Software Language Engineers' Worst Nightmare". In: *Proceedings of Software Language Engineering 2020 (SLE 2020)*. Nov. 2020. DOI: 10.1145/3426425.3426933

Lecture 7 The following papers are recommended reading only, and are not required for the annotated bibliography assignment.

- M. Hermansdörfer and G. Wachsmuth. "Evolving Software Systems". In: ed. by T. Mens, A. Serebrenik, and A. Cleve. Springer, 2014. Chap. 2. Coupled Evolution of Software Metamodels and Models, pp. 33–63. ISBN: 978-3-642-45398-4. DOI: 10.1007/978-3-642-45398-4
- M. Alanen and I. Porres. "Difference and Union of Models". In: *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications – Proceedings of the 6th International Conference, San Francisco, CA, USA, October 20–24, 2003*. Ed. by P. Stevens, J. Whittle, and G. Booch. Vol. 2863. LNCS. Springer, 2003, pp. 2–17. ISBN: 978-3-540-45221-8. DOI: 10.1007/978-3-540-45221-8_2
- R. van Rozen and T. van der Storm. "Toward Live Domain-Specific Languages: From Text Differencing to Adapting Models at Run Time". In: *Software & Systems Modeling* 18.1 (Feb. 2019). Special Section Paper on STAF2015. Received June 27th 2016. Revised May 26th 2017. Accepted June 20th 2017. First Online August 14th 2017, pp. 195–212. ISSN: 1619-1374. DOI: 10.1007/s10270-017-0608-7

Lecture 8 no additional reading

Publications related to Master and Course Projects

The following papers have resulted from student projects related to this course. These papers serve as inspirational examples only, and are not required for the annotated bibliography assignment.

- A. Hamid and V. Zaytsev. "Detecting Refactorable Clones by Slicing Program Dependence Graphs". In: *Post-proceedings of the Seventh Seminar on Advanced Techniques and Tools for Software Evolution, SATToSE 2014, L'Aquila, Italy, July 9–11, 2014*. Ed. by D. di Ruscio and V. Zaytsev. Vol. 1354. CEUR Workshop Proceedings. CEUR-WS.org, 2014, pp. 37–48. URL: http://ceur-ws.org/Vol-1354/paper-04.pdf
- J. Jansen, A. Oprescu, and M. Bruntink. "The Impact of Automated Code Quality Feedback in Programming Education". In: *Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution, SATToSE 2017, Madrid, Spain, June 7–9, 2017*. Vol. 2070. CEUR Workshop Proceedings. CEUR-WS.org, 2017. URL: http://ceur-ws.org/Vol-2070/paper-04.pdf
- N. Lodewijks. "Analysis of a Clone-and-Own Industrial Automation System: An Exploratory Study". In: *Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution, SATToSE 2017, Madrid, Spain, June 7–9, 2017*. Vol. 2070. CEUR Workshop Proceedings. CEUR-WS.org, 2017. URL: http://ceur-ws.org/Vol-2070/paper-05.pdf
- R. van Rozen and Q. Heijn. "Measuring Quality of Grammars for Procedural Level Generation". In: *Proceedings of the 13th International Conference on Foundations of Digital Games, FDG 2018, as part of the 9th Workshop on Procedural Content Generation, PCG 2018, Malmö, Sweden, August 7–10, 2018*. ACM, 2018, pp. 1–8. DOI: 10.1145/3235765.3235821
- S. Baars and S. Meester. "CodeArena: Inspecting and Improving Code Quality Metrics using Minecraft". In: *Proceedings of the 2nd International Conference on Technical Debt, TechDebt@ICSE 2019, Montreal, QC, Canada, May 26–27, 2019*. Ed. by P. Avgeriou and K. Schmid. IEEE, 2019, pp. 68–70. DOI: 10.1109/TechDebt.2019.00023

2 Assignments

Students are required to complete three obligatory practical assignment series for this course. During the first (Series 0) you work alone. This series is approved but not graded. During the second and third (Series 1 and 2) you work in the same group of two students. When you have completed the assignment you can request your lecturer to approve your work by explaining what you did. Ask your lecturer how to deliver the solutions of the assignments. Deadlines are at the end of the week. Table 3 shows how to work on assignments and Table 4 when to work on assignments and deadlines to deliver them.

Deliverable	**Type of work**
Practical Lab Series 0	Individual work
Practical Lab Series 1	Team work
Annotated Bibliography	Individual work
Practical Lab Series 2	Team work

Table 3: Assignments and how to work on them.

L	**W**	**Practical Lab**	**Writing**	**Deadline**
1	44	Series 0 and 1	Annotated Bibliography	Series 0
2	45	Series 1	Annotated Bibliography	
3	46	Series 1 Early Grading Series 1	Annotated Bibliography	
4	47	Late Grading Series 1 Series 2	Annotated Bibliography	Series 1 – Tue. Nov. 17th
5	48	Series 2	Annotated Bibliography	
6	49	Series 2		
7	50	Series 2		Series 2 – Sun. Dec. 13th
8	51	Grading Series 2		Annotated Bibliography

Table 4: When to work on assignments and deadlines to deliver them.

Next we describe the practical assignments, which include details on grading for each assignment series.

Annotated Bibliography

During the lectures and the paper sessions we use several papers in the field of software evolution. In this assignment you structure your own thoughts on these papers and exercise your skills at creating summaries and syntheses by writing a scientific paper.

Collaboration

You need to perform this assignment individually. You are allowed to discuss literature with other students, but have to write the annotated bibliography alone.

Assignment

For each paper of the **first 5 weeks** on the reading list of Section 1.6.

- **Content.** Write a concise discussion (2-4 coherent paragraphs in your own words) of the major points of the paper.
- **Format.** Submissions should use the article format, single column, standard page width, 11 point font, using the font family Times New Roman. Please use the template shown in Figure 2. All submissions should be in PDF format.
- **Page limit.** Submissions are limited to **5 pages** excluding bibliographic references. Submissions that exceed the page limit will not be graded.

The lectures provide a presentation of the reading material, but of course this is subject to the teacher's interpretations and preferences. In your text you argue your critical opinion, a perspective on the subject matter that is well-argued, insightful, and can be adopted by the reader (the audience). A good text is clear, concise, presents relevant argumentation and displays critical thinking. To further strengthen the bibliography, you can relate other literature you find yourself.

Example questions to consider are the following. What can be learnt from a paper, what is its intended audience, and what are its scientific contributions? Does the paper have practical implications, and what are the costs and benefits of applying the proposed approach or best practices (if any)? What is the research methodology, and how are its claims validated and evaluated? Are there threats to validity? How does the paper relate to other work, and to the state-of-the-art?

Hints

We encourage you to compare some annotated bibliographies and commonly used templates for choosing how to *structure* your paper, e.g., Cornell's guidelines[13]. Your paper structure might contain the following elements as discussed in the reading assignments from "Preparation Master Project".

- **Introduction.** The annotated bibliography should be a self-contained article which requires an introduction. An introduction usually describes the topic, intended audience, and sketches a structure (why, what, how).

[13] `http://guides.library.cornell.edu/annotatedbibliography`

- **Annotations per paper.** The annotations per paper might be as described in Table 5, and consider omitting irrelevant sections and include only information that is noteworthy.

Please consider the following (non-exhaustive) general writing tips.

- Write in the 'we' form. That way readers can more easily adopt a view point.
- Use the present tense where possible. This usually reduces the complexity and puts the focus on substance. Unnecessarily switching tenses can be confusing.
- Consider breaking up long sentences into shorter ones. By doing so, you can avoid bad sentences and improve the clarity of the text.
- Be concise and avoid repetition. Use your space economically.

```
\documentclass[11pt]{article}
\begin{document}
\title{Title Text}
\author{Name (and student number)\\Affiliation\\Email}
\maketitle
\section{Introduction}
The text of the paper begins here.
%your sections go here
\bibliographystyle{plain}
\bibliography{papers.bib} %create a separate file containing the BibTex
\end{document}
```

Figure 1: Annotated Bibliography LaTeX Template

Section	Description	Approx. size
Summary	Topic, problem statement, objectives and approach.	3 sentences
Significance	The subject addressed in the article is worthy of investigation.	1 sentence
Originality	The information presented was new.	1 sentence
Quality	Quality of technical content.	1 sentence
Relevance	Why and how is the paper relevant?	1 sentence
Readability	Is the paper well-structured and understandable?	1 sentence
Overall	Overall the conclusions were supported by the data.	1 sentence

Table 5: "Preparation Master Project" Suggested Annotations per Paper

Grading

The annotated bibliography will be graded using the following model:

The base grade is 7. For this grade you need to produce an annotated bibliography that conforms to the assignment described above. The factors of Table 6 modify the base grade. The grade range is 1 to 10.

Deadline

The annotated bibliography should be delivered in course week 8.

We give a deadline extension until **December 31st 2020**.

Factor	**Base grade modification**
Missing name and or introduction paragraph	-0.5
Writing quality: proper spelling, grammar, and structure.	-1.0 to +1.0
Each paper on the reading list that is missing or not covered in sufficient depth in the bibliography.	-0.5 per missing paper
The bibliography clearly argues the students critical opinion on the contents of the papers.	+0.5 to +1.0
The bibliography considers literature outside of the reading list to support argumentation.	+0.5 to +1.0

Table 6: Grading Conditions and Scoring for the Annotated Bibliography

Practical Lab Series 0 – Rascal Basics

Rascal is a meta-programming language and language workbench that enables constructing source code analyzers, programming languages, compilers and tools. We will use Rascal for the practical labs of this course.

In this lab you learn the basic facts about Rascal [14, 15, 16] and practice applying its language features. The idea is that you learn to interact with Rascal using Eclipse by doing a few small challenges in Rascal. As a reference for learning Rascal syntax, we recommend taking a look at the Rascal documentation and Rascal recipes pages.

Installation

Please see `http://www.rascal-mpl.org` for download and installation instructions. In addition to these instructions we provide the following tips.

- Make sure you obtain at most 2020-06 (nothing newer) of the RCP/RAP developers edition of Eclipse. `https://www.eclipse.org/downloads/packages/release/2020-06/r/eclipse-ide-rcp-and-rap-developers`
- Please use JDK 8. `https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html`
- We recommend using the stable version of Rascal. Use unstable only if needed.
- Adjust the `eclipse.ini` in the Eclipse folder to set the correct VM path (`-vm path`) and adjust the VM parameters (`-vmargs`), e.g., for increasing the size of the stack (`-Xms512m`) and the heap (`-Xmx4096m`). See for instance Figure 2.
- Please do not use spaces in your Eclipse path.

```
-vm
C:\Program Files\Java\jdk1.8.0_261\bin
-vmargs
-Xms512m
-Xmx4096m
```

Figure 2: Example of a partial `eclipse.ini` on Windows

Documentation

Updating Rascal's documentation with its new implementation is ongoing work.

- Rascal has built-in documentation called the Tutor.
- A copy of the Tutor can also be found online. `http://docs.rascal-mpl.org`.
- Please note that the old documentation, which is still available online, is becoming more and more outdated. `http://tutor.rascal-mpl.org/`.
- Additionally, it can help to look into Rascal's source code to learn more about its features.

Questions and Bugs Reports

Please use the following platforms for questions and bug reports.

- We invite you to pose questions and to share how to resolve issues on Slack.
- Technical questions, related to code, should be asked on Stackoverflow using the rascal tag: `http://stackoverflow.com/questions/tagged/rascal`.
- Bug reports can be submitted on GitHub. `https://github.com/usethesource/rascal/issues`.

Collaboration

Please do these exercises individually. Any communication between students is allowed, but bear in mind that you should personally be capable to code in Rascal after this.

Assignment

Fully explore the Rascal Tutor and teach yourself Rascal.

- Study its concepts, language features and library. `http://docs.rascal-mpl.org/unstable/`
- Select recipes to try out and run. You can find the sources included in the distribution. `http://docs.rascal-mpl.org/unstable/Recipes`

You will be assisted in the laboratory to install the system and type your first expressions and statements. Please ask the teachers any question about Rascal or the exercises you might have. It will be hard work!

Grading

This series is not graded. Please explore, investigate and study Rascal until you are confident you have sufficient knowledge to start Series 1.

Deadline

You should finish Series 0 in the first week of the course in order to start Series 1.

Practical Lab Series 1 – Software Metrics

In Series 1 we focus on software metrics. Software metrics are used (for example) by the Software Improvement Group (`http://www.sig.eu`) to quickly gain an overview of the quality of software systems and to pinpoint problem areas that may cause low maintainability. Some relevant questions are:

1. Which metrics are used?
2. How are these metrics computed?
3. How well do these metrics indicate what we really want to know about these systems and how can we judge that?
4. How can we improve any of the above?

In other words, you have to worry about motivation and interpretation of metrics, as well as correct implementation.

The SIG Maintainability Model provides an answer to question 1. You can read about it here:

- I. Heitlager, T. Kuipers, and J. Visser. "A Practical Model for Measuring Maintainability". In: *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*. 2007, pp. 30–39. DOI: `10.1109/QUATIC.2007.8`.
- Additional information can be found online, e.g. Baggen *et al.* [3], Visser *et al.* [29] and `https://www.sig.eu/resources/sig-models/`

The second question above ("How are these metrics computed?") is your assignment for Series 1. The third and fourth questions will be addressed during the grading session.

Collaboration

Please make groups of two students. You can work together as a pair on all aspects of this assignment. You can brainstorm with anybody else about the contents of your report, but for this assignment you are not allowed to look at code from other groups or exchange solutions in detail with other groups.

Assignment

Using Rascal, design and build a tool that calculates the SIG Maintainability Model scores for a Java project. Document your approach in a report that complements the implementation, e.g., by describing relevant design decisions, tests, results, and what you did to address threats to validity.

Calculate at least the following metrics:

- Volume,
- Unit Size,
- Unit Complexity,
- Duplication.

For all metrics you calculate the actual metric values, for Unit Size and Unit Complexity you additionally calculate a risk profile, and finally each metric gets a score based on the SIG model ($--$, $-$, o, $+$, $++$).

Calculate scores for at least the following maintainability aspects based on the SIG model:

- Maintainability (overall),
- Analysability,
- Changeability,
- Testability.

You can earn bonus points by also implementing the Test Quality metric and a score for the Stability maintainability aspect.

Use this zip file to obtain compilable versions of two Java systems (smallsql and hsqldb): zip file[14]

- **smallsql** is a small system to use for experimentation and testing. Import as-is into Eclipse and ignore build errors.
- **hsqldb** is a larger system to demonstrate scalability. Import into Eclipse. Make sure to have only `hsqldb/src` on the build path, and add the following external jars from your `eclipse/plugins/` directory: `javax.servlet_$VERSION.jar` and `org.apache.ant_$VERSION/lib/ant.jar`

Hints

- Create an Eclipse Java project with example files to test your solution on (using the Rascal test functionality).
- Create an Eclipse Java project for each of the two systems, smallsql and hsqldb too. Some few lines of code will still not compile, but commenting them out would not change the metrics too much. So commenting out just a few lines is ok in this case. It saves time!

Grading

The assignment is judged by demonstrating your results and your code to us in a small interactive session. At the end of this session you will immediately get a grade between 3 and 10.

You also have to submit a zip file containing the source code and a PDF of the obligatory report in the Canvas assignments. The files are checked for plagiarism automatically. If you worked in a team of two, submit your assignment twice: one for each student!

You will be graded using the following model. The base grade is 7. For this grade you need an implementation that conforms to the assignment described above. The implementation consists of sensible design and code. You can explain and motivate how it actually reads in the Java code and calculates the metrics

[14] `http://homepages.cwi.nl/~jurgenv/teaching/evolution1314/assignment1.zip`

Condition	Base grade modification
The metric value (total LOC) and/or score for Volume deviate without good motivation	-0.5 to -1.0
The metric value (%) and/or score for Duplication deviate without good motivation	-0.5 to -1.0
The risk profile and/or score for Unit Size deviate without good motivation	-0.5 to -1.0
The risk profile and/or score for Unit Complexity deviate without good motivation	-0.5 to -1.0
The scores calculated for the maintainability aspects deviate without good motivation	-0.5
Your tool produces output that allows easy verification of the correctness of the result (metric values, risk profiles, scores, etc. are neatly listed next to each other)	+0.5
You also implemented Test Quality and Stability and can argue their correctness	+0.5
Your tool produces correct output for hsqldb within the time span of the grading session (approximately 30 minutes); if clone detection is turned off you may get at most an extra half point	+0.5 to +1.0
You can demonstrate that your own code is of high maintainability and has proper automated tests	+0.5
You have found another metric in the literature that is not in the SIG Maintainability Model, and you can argument why and how it would improve the results	+0.5 to +1.0

Table 7: Grading Conditions and Scoring for Series 1

based on that. Your implementation can be run during the grading session on at least the smallsql project. For grading, import the smallsql project into Eclipse as-is and ignore the 100 or so build errors. Table 7 shows conditions and how they modify the grade (the teachers have a reference implementation that provides the correct outputs).

Deadline

To alleviate the planning clash with the thesis fair this year we give two options for the formative grading sessions. Students can choose between:

1. "early" grading in Week 3 on Mon. November 9th and Tue. November 10th.
2. "late" grading in Week 4 on Mon. November 16th and Tue. November 17th.

The deadline for Series 1 is extended accordingly to:
Tuesday November 17th at the latest.

Practical Lab Series 2 – Clone Detection

Code cloning is a phenomenon that is of both scientific and practical interest. In the lecture and the related papers, clone detection and management techniques were discussed, as well as the various arguments surrounding the problems that code cloning causes.

In this lab we will take a more hands-on approach by building our own clone detection and management tools. Such tools should be of help to software engineers like yourselves, so be sure that your solution will at least satisfy your own needs! Furthermore, tool building for others is a challenging activity by itself. We expect you to find some literature on this topic yourself and use it in your design. A suggested paper on visual tools for software exploration is:

- M.-A. D. Storey, F. D. Fracchia, and H. A. Müller. "Cognitive Design Elements to Support the Construction of a Mental Model during Software Exploration". In: *Journal of Systems and Software* 44.3 (1999), pp. 171–185. ISSN: 0164-1212. DOI: 10.1016/S0164-1212(98)10055-9

Compared to Lab Series 1, this assignment will be more open. Your solution will be graded using more generic criteria, with a stronger emphasis on motivation and argumentation. You will need to use literature discussed and referenced in the lectures to find and motivate good solutions: for instance for finding an appropriate clone detection algorithm.

Collaboration

Please work in groups of two students. Complete the assignment in the same groups you did for Series 1. You can brainstorm, but for this assignment you are not allowed to look at code from other groups or exchange solutions in detail with other groups.

Assignment

The assignment consists of two main deliverables:

1. A Working prototype implementation of a clone management tool, consisting of the following elements:
   (a) A clone detector (whose back-end is written in Rascal) that detects at least Type 1 clones in a Java project:
      - The detected clone classes are written to file in a textual representation.
      - Clone classes that are strictly included in others are dropped from the results (subsumption).
      - The detector is scalable and works on bigger projects such as hsqldb.
   (b) A report of cloning statistics showing at least the % of duplicated lines, number of clones, number of clone classes, biggest clone (in lines), biggest clone class, and example clones.
   (c) At least one insightful visualization of the cloning in a project. The lecture discusses several example visualizations you could use.

(d) A convincing test harness (an automated regression test framework) that ensures your clone detector works.

2. B. Design documentation that (1) describes and (2) motivates the following elements:
   (a) The 3 main requirements your tool satisfies from the perspective of a maintainer (see for instance [28]), and the related implementation choices.
   (b) The exact type of clones your tool detects. Start from Type 1, Type 2, ... but become more specific.
   (c) The core of the clone detection algorithm that you use (in pseudo-code).
   (d) The visualization(s) you implemented: how do they help a maintainer or developer?

To score a higher grade than the base grade (7), additionally do:

- Also detect Type 2 and Type 3 clone classes.
- Implement more visualizations that provide additional insight.
- Produce maintainable code that is covered by unit tests.

Implementation

Use any method you like, but go beyond what you have done in Series 1. The detection part must be in pure Rascal, the visualisation is allowed to tap into other languages. If you opt for a PDG/SDG approach, consider using a Rascal library (it is in a separate repository, clone or fork it).

Related Work on Software Clones

The following resources can help you get acquainted with clone management:

- C. K. Roy, J. R. Cordy, and R. Koschke. "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach". In: *Sci. Comput. Program.* 74.7 (May 2009), pp. 470–495. ISSN: 0167-6423. DOI: 10.1016/j.scico.2009.02.007. URL: http://dx.doi.org/10.1016/j.scico.2009.02.007
- D. Rattan, R. Bhatia, and M. Singh. "Software Clone Detection: A Systematic Review". In: *Information and Software Technology* 55.7 (July 2013), pp. 1165–1199. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2013.01.008.
- C. K. Roy, M. F. Zibran, and R. Koschke. "The Vision of Software Clone Management: Past, Present, and Future (Keynote paper)". In: *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on.* Feb. 2014, pp. 18–33. DOI: 10.1109/CSMR-WCRE.2014.6747168
- C. Kapser and M. W. Godfrey. ""Cloning Considered Harmful" Considered Harmful". In: *2006 13th Working Conference on Reverse Engineering.* Oct. 2006, pp. 19–28. DOI: 10.1109/WCRE.2006.1

The first three are overviews [25, 23, 24], the next one is a highly cited controversial piece [13], the last one is an example paper that can result from a Master's thesis – it is easy to read and contains a simplified brief overview of the field [7].

Related Work on Visualization

Use any of the following resources. Lecture slides:

- You may choose to use RASCAL's figure library. Documentation on that library can be found in Paul Klint's lecture entitled "Towards Visual Analytics"[15]

Some inspiring papers:

- H. Murakami, Y. Higo, and S. Kusumoto. "ClonePacker: A Tool for Clone Set Visualization". In: *Proceedings of the 22nd International Conference on Software Analysis, Evolution and Reengineering*. Ed. by Y.-G. Gueheneuc, B. Adams, and A. Serebrenik. IEEE, 2015, pp. 474–478. ISBN: 978-1-4799-8469-5. DOI: 10.1109/SANER.2015.7081859
- L. Voinea and A. C. Telea. "Visual Clone Analysis with SolidSDD". in: *Proceedings of the Second IEEE Working Conference on Software Visualization*. IEEE, 2014, pp. 79–82. DOI: 10.1109/VISSOFT.2014.22
- A. Hanjalic. "ClonEvol: Visualizing Software Evolution with Code Clones". In: *Proceedings of the First IEEE Working Conference on Software Visualization*. IEEE, 2013, pp. 1–4. DOI: 10.1109/VISSOFT.2013.6650525

Visualization Libraries:

- Salix is a library for interactive tools and visualizations in RASCAL using a browser[16]. Several demos are available, including live programming of state machines, similar to the running example of [27]. – powerful yet experimental
- The built-in visualization modules of RASCAL are `vis::*`. Sources[17]. Tutor[18].
- Examples of external visualisation libraries are D3[19], vis[20], Vega[21] and Gephi[22].

[15] please find it on Canvas
[16] https://github.com/cwi-swat/salix
[17] https://github.com/usethesource/rascal/tree/master/src/org/rascalmpl/library/vis
[18] http://tutor.rascal-mpl.org/Rascal/Libraries/Vis/Vis.html
[19] https://d3js.org
[20] http://visjs.org
[21] https://vega.github.io/vega/
[22] https://gephi.org

Grading

The assignment is judged by demonstrating your results and your code to us in a small interactive session. At the end of this session you will immediately get a grade between 1 and 10. Submissions that lack a report (design document) will not be graded.

You also have to drop a zip file with the source code and a PDF of the obligatory report in the assignments. The files are checked for plagiarism automatically. If you worked in a team of two, drop your assignment twice: one for each student!

You will be graded using the following model: The base grade is 7. To qualify for grading you first need a solution that complies to the assignment described above. During the grading session (15 minutes), your tool should complete on at least the `smallsql` project used in Lab Series 1.

The following conditions modify the grade: The base grade is 7. Table 8 shows grading conditions and how they modify the grade.

Condition	**Base grade modification**
Type I clone classes are incorrectly detected.	-0.5 to -1.0
Test harness (automated test framework) is unconvincing, is limited in scope or not thorough.	-0.5 to -1.0
Cloning visualization does not give insight or does not work properly.	-1.0
The clone detection report is incomplete or incomprehensible.	-0.5 to -1.0
Type II clone classes are correctly detected and visualized	+0.5 to +1.0
Type III clone classes are correctly detected and visualized	+0.5 to +1.0
Your tool and design document include additional convincing visualizations	+0.5 to +1.0
Not able to explain the core algorithm or answer questions about it	-1.0
Documentation describes unfounded, unsupported, illogical or anecdotal motivation for visualisations	-1.0
Non-interactive static visualisation	-1.0
Does not work on hsqldb	-0.5

Table 8: Grading Conditions and Scoring for Series 2

Deadline

The deadline for Series 2 is course week 7:
Sunday December 13th at the latest.

3 Change Log

Table 9 shows the changes made to this document. Contributions are by Riemer van Rozen (RvR) and Ana Oprescu (AO). The latest changes appear at the top.

V.	Date	Modification	A.
0.25	Dec 10th 2020	Clarified the scalability requirement for Series 2.	RvR
0.24	Nov 22nd 2020	Swapped the lectures of week 6 and week 7. Added optional reading on legacy software. Abbreviated bib. entries.	RvR
0.23	Nov 6th 2020	Clarified the annotated bibliography assignment.	RvR
0.22	Oct 12th 2020	Updated the schedule for course Edition 20/21. Clarified requirements for the practical lab.	RvR
0.21	Nov 7th 2019	Modified the schedule of the mini-symposium. Extended the deadline for Series 1, and added options early (week 3) and late (week 4) formative grading sessions.	RvR
0.20	Nov 4th 2019	Clarified Series 1 requires handing in a report.	RvR
0.19	Oct 7th 2019	Updated guest lecturers. Added SIG mini-symposium on Friday November 8th.	RvR
0.18	Sep 29th 2019	Updated Rascal description. Added url of the unstable Tutor. Added deadlines for 19/20. Added CodeArena paper. Increased page limit to 5 for the Annotated Bibliography assignment. Added additional resources for Series 1.	RvR
0.17	Nov 6th 2018	Modified week 4 guest lecture topic and reading.	RvR
0.16	Nov 1st 2018	Added papers written in the context of master projects. Updated bibliographical data.	RvR
0.15	Oct 28th 2018	Updated reading of lecture 4 and series 1 deadline.	RvR
0.14	Sept 17th 2018	Updated course schedule and lecturers. Added link to SIG model in Series 1. Added Salix framework to Series 2.	RvR
0.13	Nov 21st 2017	Added this week's reading.	RvR
0.12	Nov 14th 2017	Fixed a critical error that was introduced in version 0.11 of the annotated bibliography assignment description. The correct page limit is 4 pages.	RvR
0.11	Nov 5th 2017	Simplified the LaTeX template for the annotated bibliography.	RvR
0.10	Oct 29th 2017	Modified the schedule for 2017/2018.	RvR
0.09	Dec 22nd 2016	Minor clarifications and fixed typos.	RvR
0.08	Nov 30th 2016	Fixed deadlines in Table 4. Added link to Cornell guidelines on *"how to prepare an annotated bibliography"*	RvR AO
0.07	Nov 17th 2016	Added hints section to the annotated bibliography assignment. Updated the description of Series 1.	RvR AO
0.06	Nov 9th 2016	Added online test to Series 0 and added this change log.	RvR
0.01	Oct 31st 2016	Created this document based on the work of Paul Klint, Jurgen Vinju, Magiel Bruntink and Vadim Zaytsev.	RvR

Table 9: Change Log

References

[1] M. Alanen and I. Porres. "Difference and Union of Models". In: *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications – Proceedings of the 6th International Conference, San Francisco, CA, USA, October 20–24, 2003*. Ed. by P. Stevens, J. Whittle, and G. Booch. Vol. 2863. LNCS. Springer, 2003, pp. 2–17. ISBN: 978-3-540-45221-8. DOI: 10.1007/978-3-540-45221-8_2.

[2] S. Baars and S. Meester. "CodeArena: Inspecting and Improving Code Quality Metrics using Minecraft". In: *Proceedings of the 2nd International Conference on Technical Debt, TechDebt@ICSE 2019, Montreal, QC, Canada, May 26–27, 2019*. Ed. by P. Avgeriou and K. Schmid. IEEE, 2019, pp. 68–70. DOI: 10.1109/TechDebt.2019.00023.

[3] R. Baggen, J. P. Correia, K. Schill, and J. Visser. "Standardized Code Quality Benchmarking for Improving Software Maintainability". In: *Software Quality Journal* 20.2 (June 2012), pp. 287–307. ISSN: 1573-1367. DOI: 10.1007/s11219-011-9144-9.

[4] V. and Basili, G. Caldiera, and H. D. Rombach. "Encyclopedia of Software Engineering". In: Wiley, 2002. Chap. Goal Question Metric (GQM) Approach. ISBN: 9780471028956. DOI: 10.1002/0471028959.sof142.

[5] S. Erdweg, T. van der Storm, M. Völter, M. Boersma, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, G. D. P. Konat, P. J. Molina, M. Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. A. Vergu, E. Visser, K. van der Vlist, G. H. Wachsmuth, and J. van der Woning. "The State of the Art in Language Workbenches: Conclusions from the Language Workbench Challenge". In: *Software Language Engineering – Proceedings of the 6th International Conference, SLE 2013, Indianapolis, IN, USA, October 26–28, 2013*. Ed. by M. Erwig, R. F. Paige, and E. Van Wyk. Vol. 8225. LNCS. Springer, 2013, pp. 197–217. ISBN: 978-3-319-02654-1. DOI: 10.1007/978-3-319-02654-1_11.

[6] N. Fenton. "Software Measurement: A Necessary Scientific Basis". In: *IEEE Transactions on Software Engineering* 20.3 (Mar. 1994), pp. 199–206. ISSN: 0098-5589. DOI: 10.1109/32.268921.

[7] A. Hamid and V. Zaytsev. "Detecting Refactorable Clones by Slicing Program Dependence Graphs". In: *Post-proceedings of the Seventh Seminar on Advanced Techniques and Tools for Software Evolution, SATToSE 2014, L'Aquila, Italy, July 9–11, 2014*. Ed. by D. di Ruscio and V. Zaytsev. Vol. 1354. CEUR Workshop Proceedings. CEUR-WS.org, 2014, pp. 37–48. URL: http://ceur-ws.org/Vol-1354/paper-04.pdf.

[8] A. Hanjalic. "ClonEvol: Visualizing Software Evolution with Code Clones". In: *Proceedings of the First IEEE Working Conference on Software Visualization*. IEEE, 2013, pp. 1–4. DOI: 10.1109/VISSOFT.2013.6650525.

[9] I. Heitlager, T. Kuipers, and J. Visser. “A Practical Model for Measuring Maintainability”. In: *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*. 2007, pp. 30–39. DOI: 10.1109/QUATIC.2007.8.

[10] M. Hermansdörfer and G. Wachsmuth. “Evolving Software Systems”. In: ed. by T. Mens, A. Serebrenik, and A. Cleve. Springer, 2014. Chap. 2. Coupled Evolution of Software Metamodels and Models, pp. 33–63. ISBN: 978-3-642-45398-4. DOI: 10.1007/978-3-642-45398-4.

[11] I. Herraiz, D. Rodriguez, G. Robles, and J. M. Gonzalez-Barahona. “The Evolution of the Laws of Software Evolution: A Discussion Based on a Systematic Literature Review”. In: *ACM Comput. Surv.* 46.2 (Dec. 2013), pp. 1–28. ISSN: 0360-0300. DOI: 10.1145/2543581.2543595.

[12] J. Jansen, A. Oprescu, and M. Bruntink. “The Impact of Automated Code Quality Feedback in Programming Education”. In: *Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution, SATToSE 2017, Madrid, Spain, June 7–9, 2017*. Vol. 2070. CEUR Workshop Proceedings. CEUR-WS.org, 2017. URL: http://ceur-ws.org/Vol-2070/paper-04.pdf.

[13] C. Kapser and M. W. Godfrey. “"Cloning Considered Harmful" Considered Harmful”. In: *2006 13th Working Conference on Reverse Engineering*. Oct. 2006, pp. 19–28. DOI: 10.1109/WCRE.2006.1.

[14] P. Klint, T. v. d. Storm, and J. Vinju. “RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation”. In: *Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2009, Edmonton, AB, Canada, September 20–21, 2009*. IEEE, 2009, pp. 168–177. ISBN: 978-0-7695-3793-1. DOI: 10.1109/SCAM.2009.28.

[15] P. Klint, T. van der Storm, and J. Vinju. “EASY Meta-programming with Rascal”. In: *Generative and Transformational Techniques in Software Engineering III: International Summer School, GTTSE 2009, Braga, Portugal, July 6–11, 2009. Revised Papers*. Ed. by J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva. Springer, 2011, pp. 222–289. ISBN: 978-3-642-18023-1. DOI: 10.1007/978-3-642-18023-1_6.

[16] P. Klint, T. van der Storm, and J. Vinju. “Rascal, 10 Years Later”. In: *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 2019, pp. 139–139. ISBN: 978-1-7281-4937-0. DOI: 10.1109/SCAM.2019.00023.

[17] R. Koschke. “Software Evolution”. In: ed. by T. Mens and S. Demeyer. Springer, 2008. Chap. 2. Identifying and Removing Software Clones, pp. 15–36. ISBN: 978-3-540-76440-3. DOI: 10.1007/978-3-540-76440-3.

[18] D. Landman, A. Serebrenik, E. Bouwers, and J. J. Vinju. “Empirical Analysis of the Relationship between CC and SLOC in a Large Corpus of Java Methods and C Functions”. In: *Journal of Software: Evolution and Process* 28.7 (Dec. 2015). Special Issue Paper, pp. 589–618. DOI: 10.1002/smr.1760.

[19] D. Landman, A. Serebrenik, and J. J. Vinju. "Empirical Analysis of the Relationship between CC and SLOC in a Large Corpus of Java Methods". In: *30th IEEE International Conference on Software Maintenance and Evolution, ICSME 2014, Victoria, BC, Canada, September 29–October 3, 2014*. 2014, pp. 221–230. ISBN: 978-1-4799-6146-7. DOI: 10.1109/ICSME.2014.44.

[20] N. Lodewijks. "Analysis of a Clone-and-Own Industrial Automation System: An Exploratory Study". In: *Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution, SATToSE 2017, Madrid, Spain, June 7–9, 2017*. Vol. 2070. CEUR Workshop Proceedings. CEUR-WS.org, 2017. URL: http://ceur-ws.org/Vol-2070/paper-05.pdf.

[21] T. Mens. "Software Evolution". In: ed. by T. Mens and S. Demeyer. Springer, 2008. Chap. 1. Introduction and Roadmap: History and Challenges of Software Evolution, pp. 2–11. ISBN: 978-3-540-76440-3. DOI: 10.1007/978-3-540-76440-3.

[22] H. Murakami, Y. Higo, and S. Kusumoto. "ClonePacker: A Tool for Clone Set Visualization". In: *Proceedings of the 22nd International Conference on Software Analysis, Evolution and Reengineering*. Ed. by Y.-G. Gueheneuc, B. Adams, and A. Serebrenik. IEEE, 2015, pp. 474–478. ISBN: 978-1-4799-8469-5. DOI: 10.1109/SANER.2015.7081859.

[23] D. Rattan, R. Bhatia, and M. Singh. "Software Clone Detection: A Systematic Review". In: *Information and Software Technology* 55.7 (July 2013), pp. 1165–1199. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2013.01.008.

[24] C. K. Roy, M. F. Zibran, and R. Koschke. "The Vision of Software Clone Management: Past, Present, and Future (Keynote paper)". In: *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*. Feb. 2014, pp. 18–33. DOI: 10.1109/CSMR-WCRE.2014.6747168.

[25] C. K. Roy, J. R. Cordy, and R. Koschke. "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach". In: *Sci. Comput. Program.* 74.7 (May 2009), pp. 470–495. ISSN: 0167-6423. DOI: 10.1016/j.scico.2009.02.007. URL: http://dx.doi.org/10.1016/j.scico.2009.02.007.

[26] R. van Rozen and Q. Heijn. "Measuring Quality of Grammars for Procedural Level Generation". In: *Proceedings of the 13th International Conference on Foundations of Digital Games, FDG 2018, as part of the 9th Workshop on Procedural Content Generation, PCG 2018, Malmö, Sweden, August 7–10, 2018*. ACM, 2018, pp. 1–8. DOI: 10.1145/3235765.3235821.

[27] R. van Rozen and T. van der Storm. "Toward Live Domain-Specific Languages: From Text Differencing to Adapting Models at Run Time". In: *Software & Systems Modeling* 18.1 (Feb. 2019). Special Section Paper on STAF2015. Received June 27th 2016. Revised May 26th 2017. Accepted June 20th 2017. First Online August 14th 2017, pp. 195–212. ISSN: 1619-1374. DOI: 10.1007/s10270-017-0608-7.

[28] M.-A. D. Storey, F. D. Fracchia, and H. A. Müller. "Cognitive Design Elements to Support the Construction of a Mental Model during Software Exploration". In: *Journal of Systems and Software* 44.3 (1999), pp. 171–185. ISSN: 0164-1212. DOI: 10.1016/S0164-1212(98)10055-9.

[29] J. Visser, S. Rigal, R. van der Leek, P. van Eck, and G. Wijnholds. *Building Maintainable Software, Java Edition: Ten Guidelines for Future-Proof Code*. 1st ed. O'Reilly, 2016. ISBN: 9781491953525.

[30] L. Voinea and A. C. Telea. "Visual Clone Analysis with SolidSDD". In: *Proceedings of the Second IEEE Working Conference on Software Visualization*. IEEE, 2014, pp. 79–82. DOI: 10.1109/VISSOFT.2014.22.

[31] V. Zaytsev. "Software Language Engineers' Worst Nightmare". In: *Proceedings of Software Language Engineering 2020 (SLE 2020)*. Nov. 2020. DOI: 10.1145/3426425.3426933.