Software Evolution – Reader

Edition 2022/2023 – Version 0.27

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], Simon Baars (guest lecturer) [2 8], Riemer van Rozen [1], Georgia Samaritaki (lab teacher) [2], Martin Bor (lab teacher) [2], Damian Frölich (teacher) [2], and Thomas van Binsbergen (course coordinator, lecturer, lab teacher) [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
[8]Picnic

October 31, 2022

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 1.4, reading in Section 1.5 and the evaluation in Section **??**. 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, Dr. Vadim Zaytsev, and Dr. Riemer van Rozen. Current editor: Dr. L. Thomas van Binsbergen.

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 we use the 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/2021-2022/zoek-vak/vak/88818`
[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, attending lectures and working on assignments in the practical lab. All students are encouraged to work in the lab during the scheduled contact hours and to make the most out of these moments by asking for feedback from peers and teachers.

- **Reading:** 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 1.4 and weekly reading in Section 1.5.
- **Writing:** For the practical assignments technical reports are to be written that describe and reflect on the contributions made by the project as well as how the contributions are related to scientific papers. These can be papers recommended in this reader or found by the students themselves. Special attention is given to analyzing scientific papers and discussing papers as part of an annotated bibliography. The bibliography and the (other) assignments are discussed towards the end of this reader.
- **Lecture:** Throughout the course several (guest) lectures are given on topics that relate to the papers being studied or the projects being executed. These lecture hours are announced in advance, for example on the course schedule in Section 1.4. The group discusses the concepts, problems and solutions introduced during these lectures.
- **Practical Lab:** Students work on practical assignments in pairs. During contact hours, students are encouraged to ask for feedback with the teachers, improving their work over several iterations before finally handing it in before the deadline. The projects are presented and demonstrated in the final week of the course.

1.3.1 Grading

The course grade is the average of three grades[8], for practical lab *Series 1* and *Series 2* and an individual grade for an *Annotated Bibliography* of papers you have studied[9]. Grades are calculated as follows.

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

1.3.2 Submission Guidelines

When submitting your code for *Series 1* and *Series 2*

- Do not submit the files of *smallsql* and *hsqldb*
- Add your report to your project folder and zip the folder.
- Only submit 1 single compressed file

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

1.4 Course Schedule

Table 1 shows a week-by-week schedule of topics, lecture dates and lecturers. 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. The table is subject to possible changes.

Week	**Subject**	**Lecturer**	**Time**
1	Introduction to Software Evolution[22, 13]	Thomas	M 11-13
	Meta-programming and Rascal[19, 18]	Tijs van der storm	T 11-13
2	Software Metrics at SIG[12, 3]	Magiel Bruntink	M 11-12.30
3	Clone Detection and Management[20, 16]	Damian	M 11-13
4	Semantics and Equality[4, 5]	Thomas	M 11-13
5	Legacy Software and Renovation[36]	Vadim Zaytsev	M 11-13
6	Breaking Changes[25, 24, 6]	Lina Ochoa Venegas	M 11-13
7			
8			

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

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[10].

Week	**Reading**	**Practical Lab**	**Deadlines**
1	[19]. [18] [22]. [13].	Series 0 & 1	Series 0
2	[12]. [3].	Series 1	
3	[20]. [16]	Series 1	Series 1
4	[4, 5]	Series 2 & Grading Series 1	
5	[36, 35]	Series 2	
6	[25, 24, 6]	Series 2	
7	no additional reading	Series 2	Series 2
8	no additional reading	Grading Series 2	Annotated bibliography

Table 2: Course Plan: Reading, Assignments and Deadlines

[10] **Note:** Please find the exact deadlines on Canvas

1.5 Reading

This is the list of papers students read during the course, organised week-by-week the papers are covered in the lectures. Some of the papers mentioned here are mandatory for the annotated bibliography, as indicated in the description of that assignment.

Week 1

- 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
- 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
- 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

Week 2

- 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
- R. Baggen et al. "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

Week 3

- 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

Week 4 The following reading is recommended for the lectures and is not mandatory for the annotated bibliography, unless state differently elsewhere:

- B. Basten et al. "Modular language implementation in Rascal – experience report". In: *Science of Computer Programming* 114 (2015). LDTA (Language Descriptions, Tools, and Applications) Tool Challenge, pp. 7–19. ISSN: 0167-6423. DOI: 10.1016/j.scico.2015.11.003
- L. T. van Binsbergen, P. D. Mosses, and N. Sculthorpe. "Executable Component-Based Semantics". In: *Journal of Logical and Algebraic Methods in Programming* 103 (Feb. 2019), pp. 184–212. DOI: 10.1016/j.jlamp.2018.12.004

Week 5 The following reading is recommended for the lectures and is not mandatory for the annotated bibliography, unless state differently elsewhere:

- V. Zaytsev. "Software Language Engineers' Worst Nightmare". In: *Proceedings of Software Language Engineering 2020 (SLE 2020)*. Nov. 2020. DOI: 10.1145/3426425.3426933
- V. Zaytsev. "Modelling of Language Syntax and Semantics: The Case of the Assembler Compiler". In: *Journal of Object Technology* 19.2 (July 2020). Ed. by A. Vallecillo. The 16th European Conference on Modelling Foundations and Applications (ECMFA 2020), 5:1–22. ISSN: 1660-1769. DOI: 10.5381/jot.2020.19.2.a5

Week 6 The following reading is recommended for the lectures and is not mandatory for the annotated bibliography, unless state differently elsewhere:

- L. Ochoa et al. "Breaking bad? Semantic versioning and impact of breaking changes in Maven Central". In: *Empir. Softw. Eng.* 27.3 (2022), p. 61. DOI: 10.1007/s10664-021-10052-y
  A replication study of:
  - S. Raemaekers, A. van Deursen, and J. Visser. "Semantic versioning and impact of breaking changes in the Maven repository". In: *Journal of Systems and Software* 129 (2017), pp. 140–158. ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.04.008
- L. Ochoa, T. Degueule, and J. Falleri. "BreakBot: Analyzing the Impact of Breaking Changes to Assist Library Evolution". In: *2022 IEEE/ACM 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. 2022, pp. 26–30. DOI: 10.1145/3510455.3512783
- C. Bogart et al. "When and How to Make Breaking Changes: Policies and Practices in 18 Open Source Software Ecosystems". In: 30.4 (July 2021). ISSN: 1049-331X. DOI: 10.1145/3447245

Week 7 no additional reading

Week 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
- D. Frolich and L. T. van Binsbergen. "A Generic Back-End for Exploratory Programming". In: *Trends in Functional Programming: 22nd International Symposium, TFP 2021, Virtual Event, February 17–19, 2021, Revised Selected Papers*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 24–43. ISBN: 978-3-030-83977-2. DOI: 10.1007/978-3-030-83978-9_2

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
Practical Lab Series 2	Team work
Annotated Bibliography	Individual work

Table 3: Assignments and how to work on them.

Week	**Practical Lab**	**Writing**	**Deadline**
1	Series 0 and 1	Annotated Bibliography	Series 0
2	Series 1	Annotated Bibliography	
3	Series 1	Annotated Bibliography	Series 1 – see Canvas
4	Grading Series 1 Series 2	Annotated Bibliography	
5	Series 2	Annotated Bibliography	
6	Series 2	Annotated Bibliography	
7	Series 2	Annotated Bibliography	Series 2 – see Canvas
8	Grading Series 2	Annotated Bibliography	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.

Reading lists

For your annotated bibliography you will read the following mandatory papers plus a chosen extension focusing on a certain topic relevant to software evolution (you choose a list, not individual papers; no mixing allowed):

Mandatory

- 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
- 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

Choice 1: Metrics

- 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
- R. Baggen et al. "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
- 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

- T. L. Alves, C. Ypma, and J. Visser. "Deriving metric thresholds from benchmark data". In: *2010 IEEE International Conference on Software Maintenance*. 2010, pp. 1–10. DOI: 10.1109/ICSM.2010.5609747
- L. Ochoa et al. "Breaking bad? Semantic versioning and impact of breaking changes in Maven Central". In: *Empir. Softw. Eng.* 27.3 (2022), p. 61. DOI: 10.1007/s10664-021-10052-y
- L. Ochoa, T. Degueule, and J. Falleri. "BreakBot: Analyzing the Impact of Breaking Changes to Assist Library Evolution". In: *2022 IEEE/ACM 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. 2022, pp. 26–30. DOI: 10.1145/3510455.3512783

Choice 2: Software Language Engineering

- 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
- B. Basten et al. "Modular language implementation in Rascal – experience report". In: *Science of Computer Programming* 114 (2015). LDTA (Language Descriptions, Tools, and Applications) Tool Challenge, pp. 7–19. ISSN: 0167-6423. DOI: 10.1016/j.scico.2015.11.003
- 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
- V. Zaytsev. "Software Language Engineers' Worst Nightmare". In: *Proceedings of Software Language Engineering 2020 (SLE 2020)*. Nov. 2020. DOI: 10.1145/3426425.3426933
- V. Zaytsev. "Modelling of Language Syntax and Semantics: The Case of the Assembler Compiler". In: *Journal of Object Technology* 19.2 (July 2020). Ed. by A. Vallecillo. The 16th European Conference on Modelling Foundations and Applications (ECMFA 2020), 5:1–22. ISSN: 1660-1769. DOI: 10.5381/jot.2020.19.2.a5
- L. T. van Binsbergen, P. D. Mosses, and N. Sculthorpe. "Executable Component-Based Semantics". In: *Journal of Logical and Algebraic Methods in Programming* 103 (Feb. 2019), pp. 184–212. DOI: 10.1016/j.jlamp.2018.12.004

Assignment

For each paper on the selected reading list:

- **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 (i.e. do not modify the margins), 11 point font, using the font family Times New Roman. Please use the template shown in Figure 1. All submissions should be in PDF format.
- **Page limit.** Submissions are limited to **6 pages** excluding bibliographic references. Submissions that exceed the page limit will not be graded.

The lectures provide a presentation of (some of) the reading material, but of course this is subject to the teacher's interpretations and preferences. In your text you argue your own critical opinion, a perspective on the subject matter that is well-argued, insightful, and can be adopted by the reader. 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? How are these contributions evaluated? 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?

Tips

You might like to select a 'focus point' for your discussion, a theme that you return to for each of the papers, such as: practicality, impact, relation to software evolution, methodology, validity, a shared concept (such as modularity or benchmark), etc. We encourage you to compare some annotated bibliographies and commonly used templates for choosing how to *structure* your paper, e.g., Cornell's guidelines[11]. Your paper structure contains 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 (e.g., the chosen theme), intended audience, and sketches the structure.
- **Annotations per paper.** Every paper citation must have an annotation of 2-4 paragraphs as described above.

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.
- Use active voice, i.e. avoid passive voice.
- 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.

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

```
\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

Grading

The annotated bibliography will be graded using the following model:

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 5: Grading Conditions and Scoring for the Annotated Bibliography

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 5 modify the base grade. The grade range is 1 to 10.

Deadline

The annotated bibliography should be delivered in course week 8.

The precise deadline is on Canvas.

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 [19, 17, 18] and practice applying its language features. The idea is that you learn to interact with Rascal using VScode or Eclipse by doing a few small challenges in Rascal. As a reference for learning Rascal syntax, you can use the Rascal concepts page linked to below.

Documentation

Please consult the following pages as your main references for Rascal:

- Installation instructions,
- Rascal concepts,
- Rascal reference manual, and
- Rascal recipes pages

The recipes are useful example programs that show a large set of the features of the language by implementing example algorithms and small languages.

Note that both Rascal, its tooling and its documentation are under constant revision. As a consequence, bugs in the language or tooling may be encountered while working on your examples. These should be reported in the method described below. The links above are to the new documentation. The old documentation can be found here when needed. The VScode plugin is a new addition to Rascal's toolbox; traditionally Rascal programs were developed using the Eclipse plugin or the command-line shell using the .JAR provided. You might also like to take a look at the Rascal source code.

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 Rascal, 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 the exercises for Series 0 individually. Communication is allowed, but bear in mind that you should be able to program in Rascal individually after Series 0.

Assignment

Teach yourself Rascal:

- Study its concepts, language features, library and try recipes.
  `https://new.rascal-mpl.org/docs/RascalConcepts/`
  `https://new.rascal-mpl.org/docs/Recipes/`
- Solve the Series 0 problems posted on Canvas as preparation for Series 1/2.

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 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, in this assignments you concern yourself with the motivation, interpretation and implementation of metrics. 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 reading is provided by Baggen *et al.* [3], Visser *et al.* [33] and online `https://www.sig.eu/resources/sig-models/`

Question 2 is partially answered by the 2007 paper referred to above and by you in your programming for this assignment. The remaining questions are answered in the report.

Collaboration

Series 1 and 2 are executed in (the same) pairs. 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. Golden rule: "exchange ideas, not solutions!"

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 the following zip file to obtain compilable versions of two Java systems (smallsql and hsqldb): zip file[12]

- **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 a Java project with example files to test your solution on (using the Rascal test functionality).
- Create a Java project for each of the two systems, smallsql and hsqldb. 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 *grading session*. You also have to submit a zip file containing the *source code* and a PDF of the *report* in the Canvas assignment. The files are checked for plagiarism automatically.

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. Furthermore, your solution has a sensible design and code implementation. During the interactive session, you can explain and motivate how your solution reads the Java code and calculates the metrics. Your implementation can be run during the grading session on at least the smallsql project. To prepare for the grading session, import the smallsql project into Eclipse as-is and ignore the 100 or so build errors. Table 6 shows conditions and how they modify the grade (the teachers have a reference implementation that provides outputs for comparison).

[12] `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 20 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 6: Grading Conditions and Scoring for Series 1

Deadline

The precise deadline for handing in your submission is given on canvas. Shortly after the deadline the grading sessions will be held according to a schedule announced in due course.

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 build 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! 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, argumentation, evaluation and reflection. You will need to use literature discussed and referenced in the lectures to find and motivate good solutions.

Collaboration

Complete the assignment in the same group as 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. The Golden Rule still applies.

Assignment

In this assignment you will implement AST-based clone detection and use it to produce clones and statistics about clones for a given Java project (we use smallsql and hsqldb again). After this first step you will either:

- Implement clone detection algorithms to find clones of Type II and Type III **(back-end route)**
- Implement visualizations of clones (classes) that aid software maintenance **(front-end route)**

In both cases you will have to present a design of your solution (algorithms or visualisations), draw from existing literature for your design, describe the implementation of your solution, and provide a thorough evaluation and reflection on your design. These aspects are to be covered in a written report and in a presentation in the last week of the course.

The assignment consists of two main deliverables. Some parts are only required for the chosen route (where indicated):

1. Working prototype implementation of a clone management tool, consisting of the following elements:
   (a) An AST-based clone detector whose back-end is written in Rascal that detects at least Type I clones in a Java project:
      - Detected clone classes are written to a 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 (in members), and example clones.

(c) (**front-end route**) Insightful visualizations of cloning in a project that help with software maintenance. The lecture discusses several example visualizations you could use.

(d) (**back-end route**) A benchmark Java project that serves to demonstrate the correctness of your clone detection algorithms with respect to the types of clones they are meant to detect.

2. A written report that (1) describes, (2) motivates, and (3) reflects on the following elements (not in order):

   (a) An explanation of the implementation of your clone detection algorithm(s).

   (b) (**front-end route**) The requirements (at least three) your tool satisfies from the perspective of a maintainer (see for instance [32]), and the related implementation choices.

   (c) (**front-end route**) The implementation of your visualization(s) and a reflection that establishes to which extent the requirements have been satisfied by your visualizations

   (d) (**back-end route**) The exact type of clones your tool detects, giving a sufficiently detailed definition that enables critical assessment of your benchmark.

   (e) (**back-end route**) A detailed discussion of the (differences in) cloning statistics produced by your different algorithms.

To score higher grade than the base grade (7), more than one visualization and (additional) algorithm are expected in the front-end and back-end routes respectively. The additional solutions need to evaluated as thoroughly as the initial ones.

Implementation

The clone detection algorithms must be in implemented in pure Rascal. The visualisations can be written in other languages.

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
- 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

The first three are overviews [29, 27, 28], the next one is a highly cited controversial piece [16], 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 [10].

Related Work on Visualization

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[13]. Several demos are available, including live programming of state machines, similar to the running example of [31]. – powerful yet experimental
- Examples of external visualisation libraries are D3[14], vis[15], Vega[16] and Gephi[17].

[13] https://github.com/cwi-swat/salix
[14] https://d3js.org
[15] http://visjs.org
[16] https://vega.github.io/vega/
[17] https://gephi.org

2.1 Related Work on Benchmarks

- K. Jezek and J. Dietrich. "API Evolution and Compatibility: A Data Corpus and Tool Evaluation". In: *Journal of Object Technology* 16.4 (Aug. 2017), 2:1–23. ISSN: 1660-1769. DOI: 10.5381/jot.2017.16.4.a2

Grading

Series 2 is primarily assessed on the report you submit. You will also give a mandatory presentation that complements your report and can influence your grade. Your submission includes a PDF file of your report, the source-code of your implementation and statistics reports for smallsql and hsqldb. The files are checked for plagiarism automatically. You will not receive a grade if you do not support a report or the source code, or if you do not give a presentation. The presentation should describe the design of your solution(s) and details the process and results of your evaluation. The presentation is a joint presentation between both members of the group.

To qualify for grading you first need a solution that complies to the assignment as described. The base grade is 7 and the following conditions modify the grade:

Condition	**Base grade modification**
(**implementation**) Type I clone classes are incorrectly detected.	-0.5 to -1.0
(**implementation**) Missing statistics reports for smallsql or hsqldb	-0.5
(**report**) Type I algorithm is described incompletely or incomprehensibly.	-0.5 to -1.0
(**report**) Unfounded, unsupported, or illogical motivations for the design	-1.0
(**back-end route**) Benchmark is limited in scope or not thorough.	-0.5 to -1.0
(**back-end route**) Comparisons between implemented algorithms is lacking or superficial.	-0.5 to -1.0
(**front-end route**) Cloning visualizations do not give insight or do not work properly.	-0.5 to -1.0
(**front-end route**) Requirements are not clearly defined or not sufficiently reflected upon.	-0.5 to -1.0
(**front-end route**) Correct detection of Type II, III, or IV clone classes	+0.5 to +1.0
(**back-end route**) Correct detection of Type IV clone classes	+0.5 to +1.0
(**presentation**) Well-motivated design of solution(s) presented clearly	-0.5 to +0.5
(**presentation**) Sound evaluation of solution(s) presented clearly	-0.5 to +0.5
(**presentation**) Insightful discussion in response to questions	-0.5 to +0.5
You have designed, implemented, and evaluated additional solutions	+0.5 to +2.0

Table 7: Grading Conditions and Scoring for Series 2

Deadline

The deadline for handing in your submission is given on canvas. Shortly after the deadline, the presentation sessions will be held according to a schedule announced in due course.

3 Change Log

Table 8 shows the changes made to this document. Contributions are by Thomas van Binsbergen (LTvB), Riemer van Rozen (RvR), Ana Oprescu (AO) and Georgia Samaritaki (GS). The latest changes appear at the top.

V.	Date	Modification	A.
0.27	October 2022	Updated lectures and assignments for 22/23 edition.	LTvB
0.26	October 2021	Week numbers for portability. Updated deadlines. Modified texts about bibliography annotations. Other small textual updates	LTvB & GS
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 8: Change Log

References

[1] T. L. Alves, C. Ypma, and J. Visser. "Deriving metric thresholds from benchmark data". In: *2010 IEEE International Conference on Software Maintenance*. 2010, pp. 1–10. DOI: 10.1109/ICSM.2010.5609747.

[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] B. Basten, J. van den Bos, M. Hills, P. Klint, A. Lankamp, B. Lisser, A. van der Ploeg, T. van der Storm, and J. Vinju. "Modular language implementation in Rascal – experience report". In: *Science of Computer Programming* 114 (2015). LDTA (Language Descriptions, Tools, and Applications) Tool Challenge, pp. 7–19. ISSN: 0167-6423. DOI: 10.1016/j.scico.2015.11.003.

[5] L. T. van Binsbergen, P. D. Mosses, and N. Sculthorpe. "Executable Component-Based Semantics". In: *Journal of Logical and Algebraic Methods in Programming* 103 (Feb. 2019), pp. 184–212. DOI: 10.1016/j.jlamp.2018.12.004.

[6] C. Bogart, C. Kästner, J. Herbsleb, and F. Thung. "When and How to Make Breaking Changes: Policies and Practices in 18 Open Source Software Ecosystems". In: 30.4 (July 2021). ISSN: 1049-331X. DOI: 10.1145/3447245.

[7] 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.

[8] 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.

[9] D. Frolich and L. T. van Binsbergen. "A Generic Back-End for Exploratory Programming". In: *Trends in Functional Programming: 22nd International Symposium, TFP 2021, Virtual Event, February 17–19, 2021, Revised Selected Papers*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 24–43. ISBN: 978-3-030-83977-2. DOI: 10.1007/978-3-030-83978-9_2.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] K. Jezek and J. Dietrich. "API Evolution and Compatibility: A Data Corpus and Tool Evaluation". In: *Journal of Object Technology* 16.4 (Aug. 2017), 2:1–23. ISSN: 1660-1769. DOI: 10.5381/jot.2017.16.4.a2.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] L. Ochoa, T. Degueule, and J. Falleri. "BreakBot: Analyzing the Impact of Breaking Changes to Assist Library Evolution". In: *2022 IEEE/ACM 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. 2022, pp. 26–30. DOI: 10.1145/3510455.3512783.

[25] L. Ochoa, T. Degueule, J. Falleri, and J. J. Vinju. "Breaking bad? Semantic versioning and impact of breaking changes in Maven Central". In: *Empir. Softw. Eng.* 27.3 (2022), p. 61. DOI: 10.1007/s10664-021-10052-y.

[26] S. Raemaekers, A. van Deursen, and J. Visser. "Semantic versioning and impact of breaking changes in the Maven repository". In: *Journal of Systems and Software* 129 (2017), pp. 140–158. ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.04.008.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] V. Zaytsev. "Modelling of Language Syntax and Semantics: The Case of the Assembler Compiler". In: *Journal of Object Technology* 19.2 (July 2020). Ed. by A. Vallecillo. The 16th European Conference on Modelling Foundations and Applications (ECMFA 2020), 5:1–22. ISSN: 1660-1769. DOI: 10.5381/jot.2020.19.2.a5.

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