Software Evolution – Reader

Edition 2024/2025 – Version 0.29

Paul Klint [1 4], Jurgen Vinju [1 4 7], Tijs van der Storm (guest lecturer) [1 6], Magiel Bruntink [3], Davy Landman [4], Vadim Zaytsev (guest lecturer) [5], Simon Baars [2 8], Riemer van Rozen [1], Georgia Samaritaki (teacher) [2], Martin Bor [2], Damian Frölich (teacher) [2], and Thomas van Binsbergen (course coordinator) [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 25, 2024

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, and reading in Section 1.5. Section 2 describes the practical assignments. 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 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[2]. Rascal has a built-in *Tutor* that provides explanations on concepts and interactive exercises for learning to apply language features. A non-interactive version is available online[3]. Additionally, questions can be posed on Stackoverflow[4] using the `rascal` tag, and issues can be reported on GitHub[5].

[2] http://www.rascal-mpl.org
[3] http://docs.rascal-mpl.org/unstable/TutorHome
[4] http://stackoverflow.com/questions/tagged/rascal
[5] 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 (series 2) 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[6], for practical lab *Series 1* and *Series 2* and an individual grade for an *Annotated Bibliography* of papers you have studied[7]. Grades are calculated as follows.

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

There are no opportunities for resits or deadline extensions unless for extenuating circumstances to be communicated with the study advisor.

1.3.2 Submission Guidelines

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

- Do not submit the files of *smallsql* and *hsqldb*

[6]**Note:** UvA's rounding rules apply.

[7]**Note:** Practical lab Series 0 is mandatory but not graded.

- Only submit 1 single compressed file
- Check the description of the assignment for the precise contents

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[17, 9]	Thomas	M 9-11
	Introduction to Rascal[14, 13] and Series 0	Damian	M 11-13
2	Software Metrics at SIG[8, 2]	Bugra Yildiz Dennis Bijlsma	M 11-13
3	Meta-programming and Rascal[14, 13]	Tijs van der Storm	M 11-13
4	Clone Detection and Management[15, 11]	Damian	M 11-13
5	Semantics and Equality[3, 4]	Thomas	M 11-13
6	Breaking Changes[19, 18, 5]	Lina Ochoa Venegas	M 11-13
7			
8			

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

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[8].

Week	**Reading**	**Practical Lab**	**Deadlines**
1	[14]. [13] [17]. [9].	Series 0 & 1	Series 0
2	[8]. [2].	Series 1	
3	[15]. [11]	Series 1	Series 1
4	[3, 4]	Series 2	
5	[24, 23]	Series 2	
6	[19, 18, 5]	Series 2	
7	additional reading	Series 2	Series 2
8	additional reading	Presentations Series 2	Annotated bibliography

Table 2: Course Plan: Reading, Assignments and Deadlines

Software evolution involves a lot of self-study and it is important to make the most out of the contact hours that are available, especially during the lab hours.

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

1.5 Reading

This is the list of recommended papers per week of the course as relevant to the lectures of that week. Some of the papers mentioned here are mandatory for the annotated bibliography, as indicated in the description of that assignment.

Week 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`

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

- 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`

Week 4

- 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 5 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 6 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 7 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`

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: `https://dare.uva.nl/search?identifier=d0ad3c4a-5d65-44d7-bbe9-2c062598c64b`
- 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`

Recent Master Theses Related to the Software Evolution

The following links point to selected Master theses that have been published in the library of the university:

- Stefan Vlastuin (InfoSupport), 2024:
`https://scripties.uba.uva.nl/search?id=record_55066`

- Floris van Leeuwen (Software Improvement Group), 2024:
  `https://scripties.uba.uva.nl/search?id=record_55281`
- Jorrit Stutterheim (Internal), 2023:
  `https://scripties.uba.uva.nl/search?id=record_53890`
- Matthias van Ingen (Software Improvement Group), 2023:
  `https://scripties.uba.uva.nl/search?id=record_53892`
- Andreea Moise (Software Improvement Group), 2023:
  `https://scripties.uba.uva.nl/search?id=record_53889`
- Núria Bruch Tàrrega (Software Improvement Group), 2020:
  `https://scripties.uba.uva.nl/search?id=record_29378`

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. Detailed submission instructions are in this reader. Deadlines are at the very 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/Reading**	**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	Series 2	Annotated Bibliography	
5	Series 2	Annotated Bibliography	
6	Series 2	Annotated Bibliography	
7	Series 2	Annotated Bibliography	Series 2 – see Canvas
8	Presentations 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.

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, 12, 13] 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 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. After all, 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/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.* [2], Visser *et al.* [22] 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. Make sure that your report (also) answers all the questions of the above introduction.

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.

Your tool should print textual output that contains all of the above information in a way that is efficient with space, easy to read and makes it easy to confirm the calculations.

Use the following zip file to obtain compilable versions of two Java systems (smallsql and hsqldb): zip file[9]

- **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

You submit a **single** zip file containing the *source code*, a PDF of your *report*, and a *document* containing the output your tool produces for the test projects. 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

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

Condition	**Max grade modifier**
The metric value (total LOC) or ranking for **Volume** deviate without good motivation	-1.0
The metric value (%) or ranking for **Duplication** deviate without good motivation	-1.0
The risk profile or ranking for **Unit Size** deviate without good motivation	-1.0
The risk profile or ranking for **Unit Complexity** deviate without good motivation	-1.0
The scores calculated for the **maintainability aspects** deviate without good motivation	-0.5
Your report **critically reflects** on the implementation of the metrics, discusses design choices and possible alternatives	-1.0 to +1.0
Your tool produces **output** that makes it easy to reproduce and verify the (intermediate and overall) results of your analysis	-0.5 to +0.5
You have implemented **Test Quality and Stability** and can argue the correctness of your implementation	+0.5
Your tool **scales** to larger projects regarding running times as demonstrated by an analysis of the algorithmic complexity of your algorithms and/or through an empirical analysis of running the tool on projects of various sizing (including smallsql and hsqldb) in the report	+1.0
Your **code** is well-structured, modular, separates concerns, has automated tests and is easy to maintain	-0.5 to +1.0
You have found another metric in the literature that is not in the SIG Maintainability Model, can argue why and how it would improve the results, and implemented the metric.	+1.0

Table 5: Grading Conditions and Scoring for Series 1

sensible design and code implementation. In your report you explain and motivate how your solution reads the Java code and calculates the metrics, the rankings, reflects on (additional) metrics and threats to validity. Your implementation should be runnable when submitted. To demonstrate scalability empirically, also submit a *script* that makes it possible to reproduce your experiment(s) without effort. Table 5 shows conditions and how they modify the grade (the teachers have a reference implementation that provides outputs for comparison).

Deadline

The deadline for handing in your submission is given on canvas.

References

[1] 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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[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: https://dare.uva.nl/search?identifier=d0ad3c4a-5d65-44d7-bbe9-2c062598c64b.

[8] 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.

[9] 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.

[10] 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`.

[11] 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`.

[12] 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`.

[13] 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`.

[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] 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`.

[16] 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`.

[17] 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`.

[18] 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`.

[19] 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`.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] V. Zaytsev. “Software Language Engineers’ Worst Nightmare”. In: *Proceedings of Software Language Engineering 2020 (SLE 2020)*. Nov. 2020. DOI: 10.1145/3426425.3426933.