A Collaborative Approach to Teaching Software Architecture

Arie van Deursen, Maurício Aniche, Joop Aué, Rogier Slag,
Michael de Jong, Alex Nederlof, Eric Bouwers
Delft University of Technology

ABSTRACT

Teaching software architecture is hard. The topic is abstract and is best understood by experiencing it, which requires proper scale to fully grasp its complexity. Furthermore, students need to practice both technical and social skills to become good software architects. To overcome these teaching challenges, we developed the *Collaborative Software Architecture Course*. In this course, participants work together to study and document a large, open source software system of their own choice. In the process, all communication is transparent in order to foster an open learning environment, and the end-result is published as an online book to benefit the larger open source community.

We have taught this course during the past four years to classes of 50-100 students each. Our experience suggests that: (1) open source systems can be successfully used to let students gain experience with key software architecture concepts, (2) students are capable of making code contributions to the open source projects, (3) integrators (architects) from open source systems are willing to interact with students about their contributions, (4) working together on a joint book helps teams to look beyond their own work, and study the architectural descriptions produced by the other teams.

CCS Concepts

•**Applied computing** → **Collaborative learning;**

Keywords

software architecture, software engineering education, open learning, collaborative book writing.

1. INTRODUCTION

In computer science curricula, software architecture is a key component of a student's software engineering education. *Software architecture* refers to the high level structures of a software system, the discipline of creating such structures, and the documentation of these structures [18]. Documenting software architecture facilitates communication between stakeholders, captures early decisions about the high-level design, and allows reuse of design components between projects [18, 8, 2].

To support teaching software architecture, lecturers can choose from a range of text books [2, 18, 4, 21]. Nevertheless, a course on software architecture has to overcome a number of challenges:

C1 The theory of software architecture (design principles, tradeoffs, architectural patterns, product lines, etc) is often very abstract and therefore hard for a student to master.

C2 The problems of software architecture are only visible at scale, and disappear once small example systems are used.

C3 A software architect needs a combination of technical and social skills: software architecture is about communication between stakeholders, and the architect needs to be able to achieve and explain consensus.

To address these challenges, we have designed a graduate course on software architecture based on the following principles:

P1 **Embrace open source**: Students pick an open source system of choice and study its architecture. Students use it to learn how to apply architectural theories to realistic systems (C1, C2).

P2 **Embrace collaboration**: Students work in teams of four to study one system in depth (C3).

P3 **Embrace open learning**: Teams share all of their work with other students. Furthermore, students share their main result with the open source community: their architectural description is published as a chapter in an online book resulting from the course (C3).

P4 **Interact with the architects**: Students are required to offer contributions (in the form of GitHub pull requests) to the open source projects, which will expose them to feedback from actual integrators and architects of the open source projects (C1, C2, C3).

P5 **Combine breadth and depth**: Students dive deeply in the system they analyze themselves, and learn broadly from the analyses conducted and presented by other teams (C1, C3).

In this paper, we describe the resulting *Collaborative Software Architecture Course* (CSAC) which has been taught in the past four years (2013-2016) to classes of 50-100 students each.[1] We start the paper by outlining the course objectives and its contents (Section 2). We then present the results of teaching this course, covering course outcomes and student evaluations (Section 3). Furthermore,

[1]See the resulting book *Delft Students on Software Architecture* [23], the https://github.com/delftswa2016 GitHub organization, and our 2013 blogpost [22].

SIGCSE '17, March 08 - 11, 2017, Seattle, WA, USA

ACM 978-1-4503-4698-6/17/03…$15.00

DOI: http://dx.doi.org/10.1145/3017680.3017737

we discuss possibilities to the underlying course ideas to other disciplines, as well as additional ideas for further strengthening the course (Section 4). We conclude by summarizing related work and the key contributions of this paper.

2. COURSE DESIGN

2.1 Educational Objectives

The *Collaborative Software Architecture Course* aims at offering students a chance to learn and experience the concepts of designing, modeling, analyzing and evaluating software architectures. In terms of Bloom's taxonomy [3], the following educational objectives can be distinguished.

On the *knowledge* level, the course aims at enabling students to familiarize themselves with key concepts in software architecture, such as architectural views, perspectives, styles, design principles, software product lines, technical debt, and Conway's law.

On the *application* level, the course aims at enabling students to apply these theories to concrete, existing systems that are maintained by a team of people and used around the world.

On the *evaluation* level, the course aims at enabling students to assess and discuss the effect of architectural decisions made by (open source) projects. Furthermore, the course aims at enabling students to assess and discuss the relevance of certain architectural theories for a given system.

Constraints The course takes 10 weeks and is worth 5 credit points (ECTS), corresponding to $5 * 28 = 140$ hours of work per student. Each week, there are two lectures of 90 minutes each. The course is a graduate level master course for students who have completed a bachelor in computer science or related field.

2.2 Method

In order to achieve its educational objectives, CSAC adopts two central ideas. The first is to let students "adopt" an open source system. They use this system to apply and evaluate architectural theories, thus bridging the knowledge, application, and evaluation levels. To deal with the complexity of realistic systems, students work in teams of four.

The second key idea of the course is to open up all communication, so that students can learn as much as possible from each other as well as from the broader open source community. Thus, throughout the course, groups can see the work of other groups, and are encouraged to help each other. Furthermore, results from the course are shared publicly as much as possible, allowing for feedback from and interaction with the broader open source community.

On a high level, each week consists of a theoretical lecture that students apply to their own systems in the next week. In this way, each week the students describe certain aspects of the architecture of their system under study, which eventually forms the input for their book chapter.

The course follows Nick Rozanski's and Eoin Woods' book *"Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives"* [18]. Based on this book, students conduct, e.g., a stakeholder analysis, and create architectural documentation covering at least a context view, development view, architectural patterns, and an evolution perspective. Furthermore, the course covers selected additional topics in software architecture. In the past years, we have included material on architectural metrics [5], technical debt [12], the use of design sketches for communication [14], and software product lines [1]. For each of these topics, students apply theories presented to their systems under study.

The course also includes guest lectures from software architects working in industry. These lectures typically cover the role of the architect in a complex organization. Students usually do not directly apply these lessons from industry to the systems they study; instead, the guest lectures serve to illustrate how the topics covered are relevant outside the scope of open source systems as well.

In the following, we present the most important aspects of our methodology, and relate them back to the five principles P1–P5 formulated in the introduction.

Group formation and project selection (P1, P2). In the first week of the course, students themselves form groups of 4. We recommend students to form diverse groups so that they can benefit most from their varying cultural and technical backgrounds.

In addition, students must choose a project that serves as case study throughout the entire course. Students select a medium to large open source project hosted on GitHub that is still active (developers are working on it every day) and open to external contributions. Such projects typically will have several pull requests from external contributors merged per day. Furthermore, students should be confident that they are able to make a contribution to this project. Although these rules are not strict constraints, each group needs to submit their project for approval. This proposal contains the name of the project, link to the repository, and a paragraph on why they chose this project. Two different teams are not allowed to work on the same project.

To help students find a project, we provide a list of the most popular GitHub projects, extracted using GHTorrent [10]. We also suggest systems that we personally believe are interesting (in the 2016 edition, for example, we suggested Ruby on Rails, Tensorflow, or SonicPi).

The use of Git and GitHub (P2, P3). Students are required to use Git and GitHub from the start of the course. Even the course contents, schedule, and assignments are made available in a GitHub repository. We set up a dedicated GitHub organization for the course, hosting all repositories used in the course.

In the first lecture, we introduce students to Git. We add all students as collaborators to the relevant GitHub repositories. The student repositories are only available to members of the organization, and not to the outside world. Students can choose themselves to make certain results publicly available.

After a team chooses their project, we create two repositories: one empty repository to work on the assignments and the book chapter, and a public fork of their chosen project. GitHub's issues, pull requests, code review comments, milestones, and releases are used for inter and intra-team communication, and for distributing finalized assignments.

We highlight the fact that students have access to the repositories of *all* other teams. We encourage students to take a look at what other students are doing as well as what other teams have done in the past (which was already published in an online book). Students are allowed to "reuse an idea" that belongs to other groups as long as they explicitly mention it in their assignments.

Use of Slack for Communication (P2, P3). We introduce Slack[2] as a tool for students to communicate among themselves and with the teachers. Within Slack, we used different channels for various course wide discussions, announcements for important messages from the teachers, and to other technical points, such as Git.

We also created one channel per group, named after the project studied by that group. We encourage groups to use their channels for all internal group communication, as this enables the teachers

[2]http://www.slack.com.

to understand their way of working and effort. In case of questions, students can involve teachers (or other students) in their group channel simply by mentioning them in their channel. Again, all student channels are open to all students, allowing students to learn from and help each other.

Student presentations (P2, P5). As an exercise in communicating architectural decisions and trade-offs, students present their group's progress to the full class at two occasions. The first presentation is a project "pitch" around the middle of the course. In the pitch, students should present their project to other students as well as their current findings. Each group has 3 minutes of presentation plus 2 minutes of Q&A from both other students and teachers.

The second set of presentations happens at the end of the course. Each group has 15 minutes of presentation plus 10 minutes of questions. In this presentation, groups show all their findings as well as the system contributions they have made throughout the course. All presentations together usually take the entire day (from 9am to 5pm) and may be divided (depending on the number of teams) in two (parallel) sessions.

2.3 Assignments

Students face four main assignments which are part of our method as well: 1) applying theory to practice, 2) contributing to the system, 3) integrating their architectural views and perspectives into a single chapter and 4) providing feedback to other students.

Applying theory to practice (P5). After each theoretical lecture, students apply what they learned to their system. As an example, one of the first assignments is to conduct a stakeholder analysis: understanding who has an interest in the project, what their interest is, and which possibly conflicting needs exist.

To do this, the students follow the approach to identify and engage stakeholders from Rozanski and Woods [18]. They distinguish various stakeholder classes, and recommend looking for stakeholders who are most affected by architectural decisions, such as those who have to use the system, operate or manage it, develop or test it, or pay for it.

To find the stakeholders and their architectural concerns, the student teams analyze any information they can find on the web about their project. Besides documentation and mailing lists, this includes an analysis of recent issues and pull requests as posted on GitHub, in order to see what the most pressing concerns of the project at the moment are and which stakeholders play a role in these discussions and the decision to integrate a change [11].

Students deliver the results of their analysis in a readable text file via GitHub, and receive complete feedback (including the grade) from the teachers within one or two weeks after the submission. Thus, students can improve for their next deliverable.

Contributing to the system (P1, P4). As a parallel task, students contribute to the system they are analyzing. This helps them in understanding the implications of key architectural decisions, and in establishing contact with the architects and integrators of the system they study.

We do not prescribe the number of contributions each team should do. To help students, we teach them how open source development and pull requests work at GitHub. We also suggest to start with something small, such as fixing simple issues or contributing to the documentation. Some open source projects provide explicit open issues that are suitable for newcomers, which also provide a good starting point.

Writing a Book Chapter (P3, P5). Inspired by the book series covering the *"Architecture of Open Source Applications"* [6, 7], the main goal of each team is to compose a chapter describing the architecture of the system they study. At the end of the course, these chapters are bundled into a book [23].

Each group should integrate views, perspectives, assignment results, and their experience in contributing to the system into a single chapter. Each chapter should be around 5000 words, and students are encouraged to include as many diagrams or images as needed. We do not require a prescribed chapter structure, but many teams follow the views and perspectives created in the earlier assignments.

The target audience for the chapter are system stakeholders and fellow students. Students can opt to make their chapter public, implying that their chapter should appeal to a wider audience. To control quality, we make it clear to students that chapters will be published only if the group's chapter grade is higher than 7 (out of 10).

To facilitate integrating, sharing, versioning, and reviewing the various chapters and the underlying drafts, all students (and teachers) use Markdown[3] for any document they create in the course. This year, we created the final book using Gitbook[4], which offers an easy way to generate an online (HTML, EPUB, PDF) book from a GitHub repository containing Markdown sources.

Providing feedback to other students (P2, P3). The course embraces collaboration. As one of their assignments, students review a chapter from another group. We take this opportunity to teach students how scientific papers are evaluated and simulate the process with them. Using the conference management system EasyChair[5], students identify their conflicts, bid to chapters they are comfortable to review, and submit a full review of the chapter. In the end, each group needs to evaluate their received feedback and improve their chapter accordingly.

2.4 Grading

A *team grade* is based on the following items:

1. Series of intermediate deliverables corresponding to dedicated assignments on, e.g, stakeholder analysis, code metrics, particular views, or design sketches. Each partial result is evaluated using rubrics reflecting content, depth, writing, and originality. We provide the 2016 rubrics in our online appendix [24].
2. The final report (book chapter) of each team, providing the relevant architectural documentation created by the team, is graded according to the same rubrics.
3. Team presentations that were evaluated by both the teachers and students in the audience (by means of an online questionnaire).

The *individual* grades are additionally based on:

1. The personal reviews to some other group. Students that have been more critical as well as constructive receive more points.
2. Active participation in the lectures. Students consistently asking good questions or initiating useful discussions during the class receive extra points. In addition, we allow students to recommend other students that have done a great job during the lectures.

[3] https://daringfireball.net/projects/markdown/
[4] http://www.gitbook.com/
[5] https://easychair.org/

3. Their workload. Students are required to keep a weekly journal of their activities. In this journal, we expect to see which activities each student performed as well as the amount of time each one required. The effort of each student in a team should amount to the prescribed 140 hours allocated for this course.

The latter point implies that *all* students are required to make a similar time investment in this course, regardless of their background. This reflects the idea that *an architect never stops learning*.

3. RESULTS OF THE 2016 COURSE

We performed a survey with the 104 students of the 2016 edition. As the survey was optional, we obtained 48 answers (response rate of 46%). Students had to answer questions in a Likert scale from 1 (no/I don't agree at all) to 5 (yes/I completely agree). Thus, whenever we mention that students agree or believe with a statement, it means that more than half of them answered a 4 or a 5 in that question. Due to space constraints, the protocol of the survey as well as full answers and charts can be found in our online appendix [24].

3.1 Participants' profile

We have a diverse group of students when it comes to their experience. There are both students with and without industry and programming experience.

In numbers, 10% of the respondents have less than one year of programming experience, while 45% have 5 or more years of experience. 23% do not have experience in industry, 25% already have more than 3 years of experience.

The vast majority (81%) has never contributed to open source before. Half of the students claim to have good knowledge of Git, while the other half believes to know the basics.

3.2 P1: Embrace open source

Selecting an appropriate project can be difficult. Nevertheless, more than a half of the students were happy about their choice of open source project. Many said that their projects were "relevant", "fun", "interesting", and "with a welcome community".

This might explain the fact that all students were able to submit at least one pull request to their projects, and two thirds of the participants performed 1 to 3 pull requests.

Some students also believe that projects were happy with the achieved results (45.8%) and that the project was open to external contributors (58.3%).

3.3 P2: Embrace collaboration

The majority of students affirm they learned much from their own team mates. They provide varying reasons, such as the different levels of experience among team members which fostered discussions, or the learning of technical skills, such as Git and Java, from their peers. We quote a student:

> *"Everyone has something to teach, I was very happy to listen to the constructive criticism of my team mates."*

On the other hand, 3 participants did not learn enough (they chose 2 on the scale). One student indicates that there was some friction among the team members, and another complained about a team mate that did not work enough.

Concerning collaboration, Slack improves communication among team members, according to 77% of students. In addition, most students (79%) state that Slack helps them to get answers to their questions quickly, from either teachers or fellow students.

The usage of Git and GitHub (and its collaborative features such as issues and pull requests) also helps to improve student productivity. Some advantages mentioned by students are that Git and GitHub make it easier for other students to review their work. On the other hand, a few students indicate that more visual (WYSIWYG) document editors, such as Google Documents, can be better for document collaboration as opposed to the combination of Git and Markdown.

3.4 P3: Embrace open learning

Most students consider the chapter reviews they received from other students as useful, although 25% thought they were not. Students with positive feedback confirm that reviews helped them to identify flaws as well as to make the document more intuitive and interesting. Some other students indicate that reviews were superficial while others believe that the reviewer did not read the entire text.

Interestingly, most students find it useful to *write* reviews for another groups — and no student disagreed with it at all:

> *"I liked reviewing them, as it gave me the opportunity to see what other groups were doing, and giving me the opportunity to help them out."*

Watching presentations about other architectures is also considered useful by a large group of students. Negative points are frequently related to the strict and tight timing (students had 3 minutes to present their work) as well as with the lack of preparation and presentation skills of some groups (to which students had to listen).

Publishing a book at the end of the course was well received by the students: 70% of them were very proud of their chapter. In their opinion, it serves as an excellent motivational factor and inspired them to work better:

> *"It's a must have experience and you learn a lot and it brings responsibility as your work is open and public."*

3.5 P4: Interact with the architects

Most students believe that contributing to the project helped them to better understand the system they were analyzing. Only 8 students disagreed.

As teachers, we suggest students to submit a pull request before trying to talk/interview the architects. This mostly worked well: 40% of the participants believe this was a good strategy for helping them to get in contact with the senior architects of the project.

3.6 P5: Combine breadth and depth

In most lectures, students affirm that they learned much from applying the theoretical concepts to their projects. In Figure 1, we present the results for each theoretical lecture we gave in this edition. The score average is 4 (out of 5), with the exception of the variability topic, for which the median is 3. We highlight a quote from a student:

> *"I learned a lot about how the open source team approach different problems (technical debt analysis) and how the project interacts with it's environment and all involved entities around (stakeholder analysis and context view)."*

On the other hand, some students complained about difficulties in putting the theory to practice. As an example, a student thinks that some concepts are not generalizable, and thus, hard to be applied in their projects:

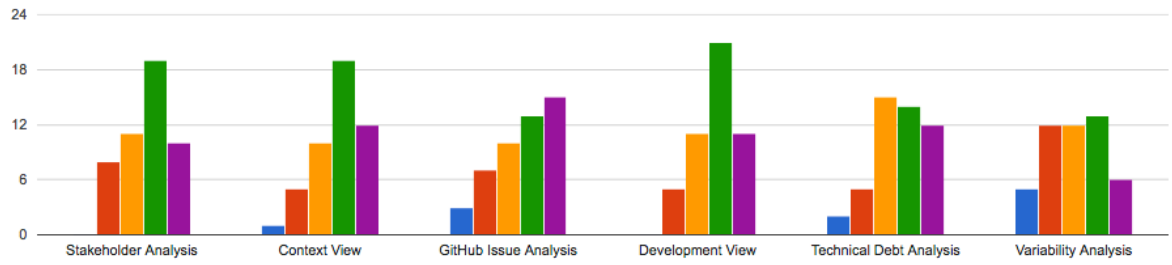

Figure 1: Histogram of student's evaluation on how much they learned in each theoretical topic that we taught in the course. Scale: 1-Blue) Useless, learned nothing from it. 2-Red) Learned something, but not enough given the effort. 3-Yellow) Neutral, learned just enough, 4-Green) Good, useful exercise, and 5-Purple) Great way to learn about this.

"It was difficult to apply the theory to our project. While I can see its value in other types of projects, it is not generally applicable."

3.7 Other findings

Time spent in the course. Students express that they spent more time in our course when compared to other courses. Only 9% of them spent the required amount of time (140 hours), while 45% spent "a lot more time". On the other hand, 19% spent around than 120 hours (less than required).

Points of improvement. Clearly, we can still make improvements to our course. At the end of the survey, we asked students about what we can improve. As an example, some students believe the course could be more technical. Indeed, our textbook treats software architecture in a highly conceptual way:

"I was hoping to focus more on architectural aspects of the software than these general exercises that just describe the application in a very broad sense."

4. DISCUSSION

As demonstrated by the above, the current mix of teaching tools and techniques works well for our course on software architecture. As one of our former teaching assistants says: "it is their chance to put hands on real applications that are not greenfield, and learn how real world works".

We believe these ideas could be extended or applied in other contexts in a number of ways:

Lectures used is a parameter of the course. The theoretical topics that we present to students during lectures can be replaced by other architectural topics of interest, such as more emphasis on design patterns or system scalability.

Mix with industry systems. Although we only made use of open source systems up to now, the course may also use projects from companies (that are most likely to be closed source). This partnership might be good for both students and companies: students can get to know more about the company, and the company can get a complete analysis of its software. On the other hand, teachers and universities have to deal with the arrangements, such as confidentiality agreements.

Collaborative book writing and publishing. This feature is clearly not attached to a software architecture course, and can thus easily be applied to any other course. As we presented before, this was one of the points which students were happy and felt motivated about. Gitbook also facilitated the generation of the final book in different versions (PDF, EPUB). Therefore, we suggest other educators to experiment collaborative book writing and publishing in their courses.

Contributions to open source. Our students were able to meet real software architects and learn from them. This relationship was initiated by these contributions and the consequent discussions (common in GitHub's pull requests) with the architects. Thus, this strategy can be used in other related courses where students could benefit from real and more experienced developers, such as software testing courses.

5. RELATED WORK

Lago and Van Vliet [13] distinguish two approaches to teaching architecture, one focusing on "programming in the large", and the other emphasizing the communication aspects of software architecture to a variety of stakeholders. Our course proposes a way to blend these two approaches in a single course.

De Boer *et al.* propose a *community of learners* approach to teaching software architecture [9]. Students collaborate on the design of a single complex system, and learn from each other. Through its openness, our course also creates a community of learners, yet student teams work on different systems.

Pedroni *et al.* [16] discuss leveraging open source projects to expose students to real life systems. As in our course, they require students to make contributions to open source projects. The course focuses on programming skills as well as on the need to get socially involved with other developers. The authors recommend providing clear instructions on how to contribute – which we indeed cover in the lectures of our course, and which these days are often also provided in contribution guidelines of projects on, e.g., GitHub. Smith *et al.* [19] discuss challenges and guidelines for selecting open source projects for use in software engineering education. Marmorstein [15] discusses experiences in letting students contribute to open source systems in their class project.

GitHub plays a central role in our course: The teams use it to collaborate, to write their book chapter, and to contribute to open source projects. This emerging role of the GitHub platform as a general collaborative tool in education is further discussed by Zagalsky *et al* [25].

Our student based book series [23] was directly inspired by the *Architecture of Open Source Applications* [6, 7] initiated by Brown and Wilson. Based on these books, Robillard and Medvidovic provide an analysis of the dissemination processes in open source architecures [17]. A description of the architectural beauty of (open source) systems was provided by Spinellis and Gousios [20].

6. CONCLUSIONS

Teaching software architecture should be practical and challenging at the same time. Towards this goal, we propose a course structure that follows five main principles: embrace open source, embrace collaboration, embrace open learning, interact with the architects, and combine breadth and depth.

We have applied these ideas in four editions of our Software Architecture course, and students' feedback have always been positive. In this paper, we report the results of the evaluation with our students in the most recent (2016) edition.

Our experience suggests that (1) open source systems can be successfully used to let students gain experience with key software architecture concepts; (2) students are capable of making meaningful code contributions to the open source projects; (3) software architects from open source systems are willing to interact with students about their contributions; (4) working together on a joint book helps teams to look beyond their own work, and study the architectural descriptions produced by the other teams.

Thanks to the open nature, results of the course (such as the online book, and contributions to the open source systems made by the students) are available in our online appendix [24]. Based on a blog post covering the first edition of CSAC [22], similar courses have emerged at various universities in Canada, Israel, France, and US. Moreover, we anticipate that our collaborative approach makes sense not only for software architecture courses, but to any other topic in which practice and theory should walk together.

Acknowledgments We would like to thank Felienne Hermans and Nicolas Dintzner (TU Delft) for repeatedly offering guest lectures in this course, the various guest speakers from industry and academia, all students participating in the courses, and the open source developers who welcomed our student's contributions.

7. REFERENCES

[1] S. Apel, D. S. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer, 2013.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture In Practice*. Addison-Wesley, third edition, 2012.

[3] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl. Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain. *Longmans, Green*, 1956.

[4] J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.

[5] E. Bouwers. *Metric-based Evaluation of Implemented Software Architectures*. PhD thesis, Delft University of Technology, 2013.

[6] A. Brown and G. Wilson. *The Architecture of Open Source Applications: Elegance, Evolution, and Fearless Hacks*, volume I. http://aosabook.org, 2012.

[7] A. Brown and G. Wilson. *The Architecture of Open Source Applications: Structure, Scale, and a Few More Fearless Hacks*, volume II. http://aosabook.org, 2013.

[8] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2010.

[9] R. C. de Boer, R. Farenhorst, and H. van Vliet. A community of learners approach to software architecture education. In *22nd Conference on Software Engineering Education and Training*. IEEE Computer Society, 2009.

[10] G. Gousios. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[11] G. Gousios, A. Zaidman, M. D. Storey, and A. van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *37th IEEE/ACM International Conference on Software Engineering, ICSE*, pages 358–368. IEEE Computer Society, 2015.

[12] P. Kruchten, R. L. Nord, and I. Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21, 2012.

[13] P. Lago and H. van Vliet. Teaching a course on software architecture. In *18th Conference on Software Engineering Education and Training*. IEEE Computer Society, 2005.

[14] N. Mangano, T. D. LaToza, M. Petre, and A. van der Hoek. How software designers interact with sketches at the whiteboard. *IEEE Trans. Software Eng.*, 41:135–156, 2015.

[15] R. M. Marmorstein. Open source contribution as an effective software engineering class project. In *Proceedings of the 16th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2011*, pages 268–272. ACM, 2011.

[16] M. Pedroni, T. G. Bay, M. Oriol, and A. Pedroni. Open source projects in programming courses. In *SIGCSE 2007*, pages 454–458. ACM, 2007.

[17] M. P. Robillard and N. Medvidovic. Disseminating architectural knowledge on open-source projects. In *Proceedings of the 38th ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 476–487. ACM, 2016.

[18] N. Rozanski and E. Woods. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2012.

[19] T. M. Smith, R. McCartney, S. S. Gokhale, and L. C. Kaczmarczyk. Selecting open source software projects to teach software engineering. In *The 45th ACM Technical Symposium on Computer Science Education, SIGCSE'14*, pages 397–402. ACM, 2014.

[20] D. Spinellis and G. Gousios. *Beautiful Architecture: Leading Thinkers Reveal the Hidden Beauty in Software Design*. O'Reilly, 2009.

[21] R. N. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Addison-Wesley, 2010.

[22] A. van Deursen. Teaching software architecture: With GitHub! https://avandeursen.com, December 2013.

[23] A. van Deursen, M. Aniche, and J. Aué. *Delft Students on Software Architecture: DESOSA 2016*. Delft University of Technology, 2016. https://www.gitbook.com/book/delftswa/desosa2016/details.

[24] A. van Deursen, M. Aniche, J. Aué, R. Slag, M. de Jong, A. Nederlof, and E. Bouwers. A Collaborative Approach to Teaching Software Architecture (SIGCSE 2017 Appendix). https://doi.org/10.5281/zenodo.182614, Nov. 2016.

[25] A. Zagalsky, J. Feliciano, M. D. Storey, Y. Zhao, and W. Wang. The emergence of github as a collaborative platform for education. In *Proceedings of the 18th ACM Conf. on Computer Supported Cooperative Work & Social Computing, CSCW 2015*, pages 1906–1917. ACM, 2015.