Quantitative Analysis of Software Designs: Teaching Design and Experiences

Alireza Hakamian
University of Stuttgart
Stuttgart, Germany
mir-alireza.hakamian@iste.uni-stuttgart.de

Steffen Becker
University of Stuttgart
Stuttgart, Germany
steffen.becker@iste.uni-stuttgart.de

ABSTRACT

Context. The Software Quality and Architecture group (SQA) at the University of Stuttgart offers the Quantitative Analysis of Software Designs (QASD) course for master students. The goal is to give students the necessary skill to evaluate architecture alternatives of software systems quantitatively. The course offers a combination of required theoretical skills, such as applying stochastic processes and practical exercises using suitable tools. *The challenge.* is providing teaching materials that balance necessary theoretical knowledge and appropriate tooling that can be used in practice. *As a solution*, the course is designed so that one-third is about the formalisms behind quantitative analysis, including stochastic processes and queuing theory. One-third is modeling languages, such as queuing networks, UML, and UML profiles, including MARTE. The other one-third uses tooling to model and analyze example systems. During Corona, we provided students with an e-learning module with pre-recorded videos, online quizzes at the end of every chapter, and a virtual machine that pre-installed all the required tooling for the exercise sheets. *Final-remarks.* In the past two years, students' feedback was often positive regarding the balance between theory and tooling. However, it has to be emphasized that the number of students participating in the course has always been no more than ten. Hence, the student feedback has not been collected by the universities' survey.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**; **Software design tradeoffs**.

KEYWORDS

teaching, software engineering, quality analysis, UML profiles, formal methods

ACM Reference Format:
Alireza Hakamian and Steffen Becker. 2023. Quantitative Analysis of Software Designs: Teaching Design and Experiences. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3578245.3584357

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04…$15.00
https://doi.org/10.1145/3578245.3584357

1 INTRODUCTION

Quality prediction of software systems during the development phase has proven to be cost-effective in the long term. Hence, a required skill for future software engineers. The Software Quality and Architecture (SQA) group at the University of Stuttgart offers Quantitative Analysis of Software Design (QASD) aiming to provide students with the required skills for (1) quality prediction, including performance, reliability, and safety, and (2) quality trade-off analysis.

The main challenge is to provide a balance between theoretical skills and Software Engineering (SE) practices. Quality prediction requires creating analytical models and solving them, which requires theoretical skills, such as knowledge of stochastic processes. On the other hand, in practice, development teams often prefer to work with high-level languages such as Unified Modeling Language (UML) [8] to describe the system's behavior and structure.

This paper aims to provide insight into the QASD course organization, including lectures, exercises, online videos, quizzes, and the final exam. Furthermore, the paper discusses the rationale behind the teaching designs and experiences of conducting the course during the pandemic.

The course teaches existing methods and related tooling for predicting software design performance, reliability, and safety. The course is organized into 11 chapters. To balance between theories and SE practice, one-third of the chapters describe formalisms, such as Markov processes, and the other third include SE practices for performance and reliability modeling through Stereotyping. The remaining is about using tooling to create and solve models that capture the qualities of the system. Furthermore, we explain how analytical models can be generated from quality annotated models. Although, not in detail.

During the corona pandemic, we adopted the flipped classroom model and decided to organize the course on an e-learning platform provided by the university. To support students in dealing with installation issues of tools, we provided a virtual machine with all the required tools. Furthermore, we offered a few online Q&A sessions and video recordings for all lecture contents. One reason for video recording is that teaching in a synchronous online session is often exhausting for students, mainly when it goes beyond 45 minutes. Hence, students can watch the videos at their own pace. In addition to Q&A sessions, we had synchronous online sessions for all exercises where we discuss possible solutions. We could not run a standard university survey to get feedback as the number of participants has been consistently no more than ten. However, through discussions, students were happy about the balance between theory and application in SE.

2 OVERVIEW OF QASD COURSE

This section gives an overview of the course organization concerning the content.

2.1 Literature

Three main books cover a large extent of the lecture materials, including Model-based software performance analysis [5], the Palladio approach in modeling and simulating software architectures [10] and the application of Queuing networks and Markov chain in performance evaluation [4]. Moreover, the literature contains, e.g., websites for tools such as *PRISM* [7] and specifications of standards such as UML.

2.2 Lectures

The lecture contains 11 chapters, each with its own dedicated exercise sheet. The following gives an overview of each chapter.

(1) **Introduction** gives motivation to how the course improves students' skills and how the course is organized
(2) **Modeling** where the content centers around describing software system architecture through UML. The aim is to refresh students' memory on the representation of architecture models such as component, sequence, and activity diagrams
(3) **UML Profiles** discuss SPT and MARTE, which are means to annotate UML models for quality predictions
(4) **Discrete-time Markov chain (DTMC)** is the first chapter concentrating on formalism required for reliability analysis
(5) **Continuous-time Markov chain (CTMC)** offers the theoretical background required for performance analysis
(6) **Queues and Queuing Network** discusses how queues are built on top of *CTMC* for performance analysis
(7) **Running Example** presents an end-to-end example for performance and reliability analysis. The goal is to manually transform UML models to queuing network and DTMC to perform performance and reliability analysis
(8) **Palladio Component Model (PCM)** chapter presents a model-driven approach for designing architectural models and Performance prediction using simulation
(9) **Safety** lecture gives an overview of quantitative and qualitative hazard identification and analysis methods, such as Fault-tree
(10) **Real-Time Systems** presents means for verifying system properties that include timing guarantees
(11) **Trade-off Analysis** In the last chapter, we discuss methods for analyzing optimal design alternatives concerning software systems that must satisfy more than one quality attributes that conflict with each other such as performance, reliability, and cost

The following Section describes the exercises and presents an end-to-end example, illustrating theoretical learning and software engineering practices together for software system quality prediction.

2.3 Exercises: End-to-End Example

Students are supposed to form a group of two to three members and work together. The seventh exercise sheet is the running example aiming to assess learning materials for UML modeling, profiles, queuing network, DTMC, and CTMC. The task presents an imaginary e-commerce web application modeled with a sequence and deployment diagrams that are annotated with performance characteristics through MARTE.

Figure 1 illustrates interactions among three components, *OrderManagment*, *Inventory* and *Database*. Furthermore, the figure shows annotations that enrich the model with performance characteristics using the MARTE profile. Figure 2 illustrates the deployment view of the web application annotated with the MARTE profile.

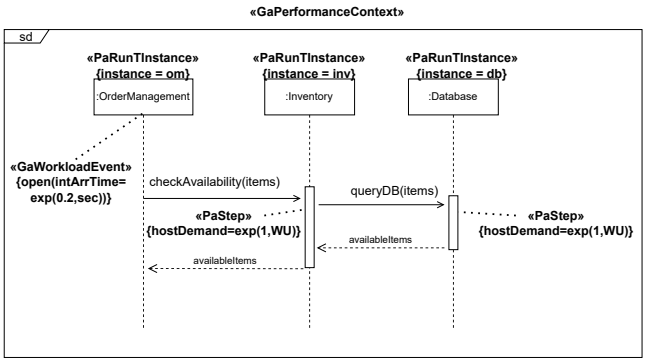

Figure 1: Sequence Diagram

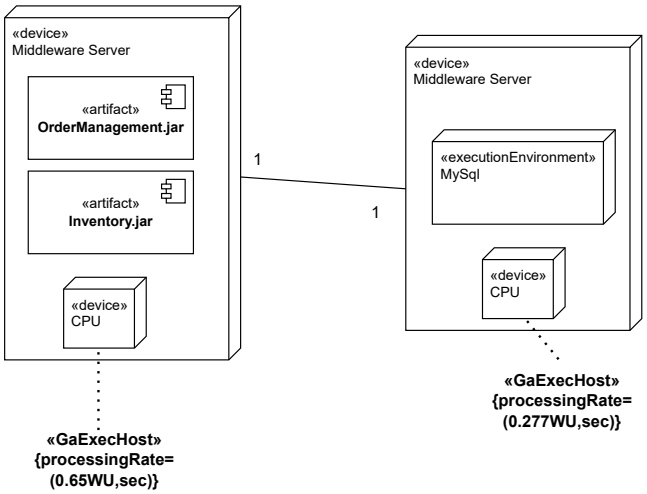

Figure 2: Deployment Diagram

The exercise asks students to use *Jsimgraph* from Java Modeling and Tools *(JMT)* [3] to draw a queuing model that reflects the UML diagrams and, through both simulation, and analytical solution predict the utilization of the *Middleware* and *Database* servers.

Figure 3 shows the resulting queuing network. According to the annotations, The mean arrival rate to the system is $\lambda = 0.2$, and the mean service rate that the servers offer are $\mu = 0.69$ and $\mu = 0.227$, respectively. After simulating the model, the utilization of *Middleware* server is 29%, and *Database* server is 79%. Because

of the high utilization of the database server, we ask students to design and simulate two alternative designs, including (1) adding a new powerful processor that reduces the mean service time to 2.7 seconds, which means the mean service rate is $\frac{1}{\mu} = 0.37$, and (2) adding cache to *Middleware* server, which according to the measurements only 20% of requests results in cache hits. The design alternative one shows utilization is reduced to 53% while the cache option leads to 57% utilization of the database server.

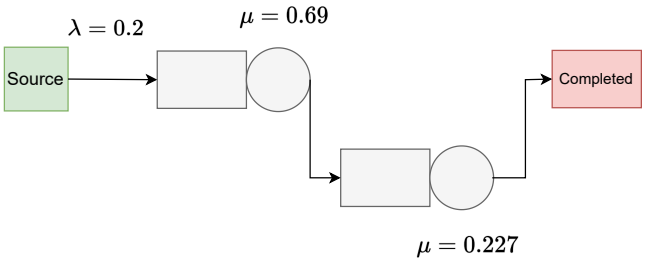

Figure 3: Queuing Network

Furthermore, the exercise asks to conduct reliability analysis using *PRISM* [7] tool and *DTMC* formalism. The task informs students that the failure probability of *Middleware* server is 0.5% in all cases, and the database fails with the probability of 0.2% in all cases. Figure4 illustrates the state space of modeled DTMC in *PRISM*. The reliability property reads as *what is the probability that the model reaches the failed state in the steady state*. The property looks similar to P=? [$F\ state = failed$]. Intuitively, the question asks with what probability the requests from *OrderManagment* component fail. Through model-checking, the result is 0.699%. Repeating the task of adding a new processor with a failure rate of 0.0028 yields a 0.77% probability of reaching failed state, and with cache, the probability of failure on demand is 0.65%.

Often it is interesting for students to see how the use of formalisms together with modeling techniques developed in SE and tooling support software engineers in quantitatively evaluating different alternatives of software designs. The results from this exercise are used for alternative analysis in the last chapter. The final chapter introduces the analytic hierarchy process (AHP) to allow students to find which of the design alternatives are optimal concerning which quality is more important.

2.4 Exam

While we had one instance of a written exam, we decided to switch to an oral exam. The main reason is that the number of students is too few to justify creating, supervising, and correcting written exams. In the exam, the focus is more on SE rather than exercising theoretical background. Whenever needed, we provide definitions of required theories and statistical concepts. The aim is to check if students learned how to use them to assess the quality and not whether they remembered definitions.

The following Section discusses the course organization, influenced by the Corona pandemic.

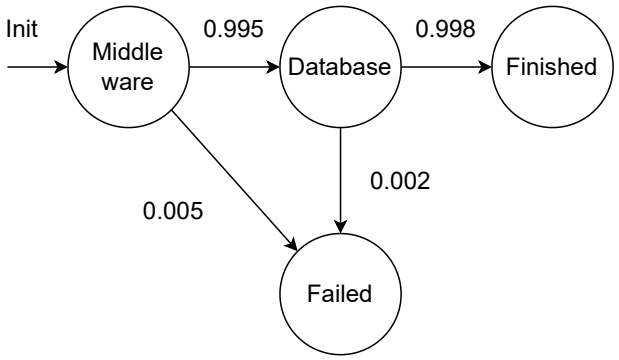

Figure 4: DTMC

3 ONLINE TEACHING EXPERIENCE

We decided to switch to the flipped classroom model such that more time is dedicated to Q&A and exercising lecture materials rather than teaching. For the teaching part, we provided students with video recordings and organized the course on University's e-learning platform.

3.1 E-Learning Module and Quizzes

The decision was to make short videos of no more than 20 minutes to prevent loss of attention. For each chapter, some Quizzes need to be passed by students to unlock the next chapter's recordings. We held weekly exercises where one group presented and discussed the solution. We held Q&A round online based on the general topics. For example, after completing UML modeling and the profiles, we offered an online Q&A session.

3.2 Virtual Machine

Supporting students in dealing with tool installation issues in an online course is challenging. We decided to compile a virtual machine with already installed needed tools for exercises, including *Papyrus* [9], *PRISM* [7], *JMT* [3], *PCM* [1], *Octave* [6], and *UPPAAL* [2].

3.3 Learning Track Issue

Often an important activity during learning for students is communicating with each other. One of the main reasons is that they know how differently they understood the topic relative to other students. Sometimes, a topic may be complex, and students with less communication with others assume the difficulty is only for them. That often results in overwhelming thoughts and reduces their learning progress. While, in general, this is an issue with no easy workaround, we think that pushing students to work as a group and discuss their solution could increase the chance of communication among them.

4 CONCLUSION

This paper summarizes QASD teaching design and experiences. The course aims to teach students the necessary skills for both quality prediction and quality trade-off analysis. Consequently, the

design of the lecture includes a balance of (1) formalisms and theories behind quantitative analysis, (2) modeling languages such as UML, and (3) tooling support for modeling and analysis activities. The paper presents and solves an end-to-end example from QASD exercise aiming to assess learning materials, including UML modeling and annotations, DTMC, CTMC, queuing network, and related tooling. Furthermore, the paper discusses the changes in the lecture settings during the pandemic, including the e-learning module with recorded videos and quizzes and supporting students regarding tooling by providing them with a virtual machine with all installed tools.

REFERENCES

[1] Steffen Becker, Heiko Koziolek, and Ralf H. Reussner. 2009. The Palladio component model for model-driven performance prediction. *J. Syst. Softw.* 82, 1 (2009), 3–22. https://doi.org/10.1016/j.jss.2008.03.066

[2] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. 2006. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA*. IEEE Computer Society, 125–126. https://doi.org/10.1109/QEST.2006.59

[3] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. 2009. JMT: performance engineering tools for system modeling. *SIGMETRICS Perform. Evaluation Rev.* 36, 4 (2009), 10–15. https://doi.org/10.1145/1530873.1530877

[4] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. 2006. *Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications, Second Edition.* Wiley. http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471565253.html

[5] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. 2011. *Model-Based Software Performance Analysis.* Springer. https://doi.org/10.1007/978-3-642-13621-4

[6] GNU Octave. 2023. GNU Octave Scientific Programming Language. https://octave.org/

[7] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2002. PRISM: Probabilistic Symbolic Model Checker. In *Computer Performance Evaluation, Modelling Techniques and Tools 12th International Conference, TOOLS 2002, London, UK, April 14-17, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2324)*, Tony Field, Peter G. Harrison, Jeremy T. Bradley, and Uli Harder (Eds.). Springer, 200–204. https://doi.org/10.1007/3-540-46029-2_13

[8] OMG Standards Development Organization. 2023. Unified Modeling Language Specification. https://www.omg.org/spec/UML/2.5.1/About-UML/

[9] Papyrus. 2023. Eclipse Papyrus Modeling Environment. https://www.eclipse.org/papyrus/

[10] Ralf H Reussner, Steffen Becker, Jens Happe, Robert Heinrich, and Anne Koziolek. 2016. *Modeling and simulating software architectures: The Palladio approach.* MIT Press.