Stefan Winter

Teaching Statement

During my Ph.D. studies and my postdoc at TU Darmstadt I have constantly been involved in organizing and teaching courses of different formats. I enjoy working with students and the most successful courses we organized were those in which we had close interactions with students and gave hands-on training. Scaling courses with close interactions to large numbers of students is challenging, but we managed to expand this to course sizes of up to 200 students.

Teaching Experience

I have experience teaching courses on both the graduate and undergraduate level and in both German and English.
Betriebssysteme: On the undergraduate level, I have developed and taught a basic course on operating systems in German, covering fundamental operating system concepts like processes, memory management, file systems, etc., with small excursions into advanced or recent topics like the Meltdown vulnerability and the implications of its fix on system performance. The course was designed as an annual 5 CP course with weekly lectures and bi-weekly exercise sessions and attracted up to 600 participants from the computer science and related study programs.
Operating Systems: The graduate level counter part of the undergraduate level operating system course was designed as an annual 8 CP course in English. As the course was popular among international students who obtained their Bachelor degrees from other universities and had not attended the undergraduate operating system course at TU Darmstadt, the course first provided a recap of the basic material from the undergrad course in English and then proceeded with advanced topics like real-time scheduling, different virtualization techniques, etc., with weekly lectures and weekly exercise sessions. The course also featured hands-on labs, in which the students learned basic system-level programming and extended Linux with simple functionalities via kernel modules. The course attracted 100-200 students.
Reliable Software and Operating Systems: The annual 8 CP course was offered as an in-depth extension of reliability aspects that we only briefly covered in the graduate level operating system course. The main thrust of the course was on testing, verification, and fault-tolerance techniques for system code, including fault injection tests, symbolic execution and concolic testing with KLEE, bounded model checking, but also checkpointing strategies, and N-version programming. The weekly lectures and exercises were augmented with hands-on small-scale projects, in which students worked with testing and analysis tools that implemented the concepts taught in class.
Seminars: In addition to the above courses we organized research seminars each semester to discuss recent research articles with students. Depending on the colleagues involved, these seminars would cover either systems topics (mostly related to operating system design or system reliability), software engineering topics (mostly related to testing) or a combination of both with interesting intersections, such as the proper design of distributed build continuous integration systems. The research seminars followed a conference simulation model, i.e., after their topic assignments students performed literature research, wrote a critical treatise, submitted that via a HotCRP instance, peer reviewed

other students' submissions, and improved their final submissions. At the end of the semester, the results were presented in a mini-conference.

Mentoring

To date I had the pleasure to mentor 35 students working on their Bachelor or Master theses as well as 3 PhD students (detailed list in my CV). There is no dedicated course or training on research methods at TU Darmstadt and students often had trouble approaching their theses. Over the years, I have collected a good amount of material covering the problems that I saw students struggle with and I would be happy to transform all this experience in an actual course that a broader audience could benefit from.

Teaching Interests

Based on my experience I am qualified to teach courses both on the graduate and undergraduate level in both German and English and for various numbers of students. I can prepare quickly for courses related to software engineering, programming, and systems, but can cover other basic courses as well with a bit of preparation.
Besides the aforementioned course on research methods, I can think of three courses to complement the study program on the graduate level.
Softwaretest und -qualitätssicherung (4h): While virtually all universities (luckily) offer a corresponding course, I would like to offer this as an addition that I myself took as an undergraduate student at TU Darmstadt. The course covered basic topics in software testing and test management and prepared students for an ISTQB certificate test that students could take at a reduced rate at the end of the semester. I personally perceived this as an attractive option, as it provided an additional qualification for the job market and got me so interested in testing that finally academia became my job market. I took the course in German, but the teaching materials (and likely the tests as well) are available in English.
Software Engineering for Mission-Critical Systems (4h): In my research I have worked with systems for which development and assessment standards exist that do not apply for commodity software. Given the strong industrial applications focus, teaching the corresponding standards' requirements and the advocated techniques would be beneficial for German students and provide an opportunity to view common software engineering and programming practices from a different angle and with a different trade-off between ease/speed of implementation and reliability.
Complex software systems analysis (4h): I have worked with a number of large and complex software systems as well as test data collections and I had to adjust my own tool set for this work over the years to scale with the targeted software and data. The goal of this course would be to sensitize students for the computational complexity that analyzing complex software systems entails and to teach them techniques to cope with this complexity, such as proper use of parallel compute resources and clusters, memory and storage management, and the related programming models.