Analyzing Software Based on Real Execution Data

In today's world, we increasingly rely on information technology. We text with each other via our smartphones, we shop online, our trains are controlled via software, and also our factories are increasingly automated. Such complex software-driven systems are supporting and automating all kinds of tasks and can be found in all sectors, ranging from communication and production to education and entertainment. In this thesis, we aim to understand and analyze these software systems using process mining techniques. That is, we see software systems as complex processes which we can analyze using recorded observations known as execution data. The idea is that we can use process execution data to gain insights about the real execution of these processes. So far, process mining techniques have had great success in understanding and analyzing organizational/business processes, but there has been little work on using process mining on software systems.

In this thesis, we addressed research challenges in using process mining techniques for analyzing software. First of all, we looked into obtaining execution data from running software. How can we record such data, what does this data look like, and what are the various properties of such data? Next, we looked into using execution data for process discovery. In process discovery, we automatically discover a process model from the recorded execution data. Such a discovered process model explains what tasks or activities actually happened in which order in the observed/recorded process. In addition, we investigated how we can use discovered models together with the recorded execution data to analyze various additional perspectives. We looked at which parts of the software are executed more frequently, where various bottlenecks or other performance issues are, and more. Finally, we investigated various ways of showing such models and results to end users.

We motivated our contributions based on detailed discussion software execution data and the associated challenges. We recognized that 1) typical software behavior is large, complex, and contains some form of hierarchical or layered structure (hierarchical and recursive behavior) and 2) that software can produce errors (cancelation behavior). Therefore, we developed modeling notations, discovery techniques, analysis techniques, and model translations supporting hierarchy, recursion, and cancelation behavior. To make the ideas and theories in this thesis usable, we developed and made available extensive tool support for round-trip software analysis. This tool support enables analysts and developers to start right away on their own software source code, analyze its runtime behavior, and link the analysis results directly back to their source code.

All methods have been implemented in various tools (in the context of ProM and XES), which are publicly available with documentation. In addition, all techniques have been systematically evaluated and have been applied in real-life situations in the context of several case studies including an industrial case study at high-tech company ASML. We demonstrated how human (domain) knowledge and software process mining results can be combined to understand and analyze legacy software, i.e. existing software for which the available knowledge and documentation is lacking, outdated or incomplete. This way we showed how our approach can aid in reverse engineering and re-engineering efforts targeted at managing legacy software.