Higher-Order Process Engineering
in the context of Active Continuous Quality Control

Johannes Neubauer
Computer Science, Chair for Programming Systems
TU Dortmund, Germany
johannes.neubauer@cs.tu-dortmund.de

Abstract: In this talk we present how Higher-Order Process Engineering (HOPE) and Active Continuous Quality Control (ACQC) can be combined to drastically reduce the manual effort of risk-based regression testing. That is, integrating active automata learning, for automatically maintaining test models of a system under test (SUT), with a rigorous model-driven quality assurance process.

Today’s enterprise software systems are increasingly based on complex stacks of technologies and the integration of heterogeneous third party components. Especially long-running multi-user systems like web applications possess this heterogeneity, which makes it particularly difficult if not impossible to control/predict the overall behavior at the system level and therefore calls for advanced system-level testing. In real-life, the situation is even more problematic, because the systems evolve at an increasing pace. Even worse, each of these changes typically requires substantial manual effort in order to have an up-to-date testing harness. These requirements are infeasible in most practical scenarios, both for cost and time. In order to tackle this issue, risk-based testing [FR14] has been introduced. It focuses on detecting the most critical bugs. This is achieved by test prioritization based on the results of a previous risk analysis.

In this talk we present how Higher-Order Process Engineering (HOPE) [NS13] and the active automata learning based Active Continuous Quality Control (ACQC) [WNS+13] approach can be combined to drastically reduce the manual effort of risk-based regression testing. Automata learning [SHM11], sometimes referred to as test-based modeling, provides the benefits of model-based testing, but does not require any a priori models. Our model-driven approach has several dimensions (cf. Fig. 1):

- **Symbol Models** reflect single user actions. They are generic in that they may have multiple input and output parameters. Symbol models are used to define generic learning alphabet symbols based on test blocks. Test blocks provide a stable abstraction [WNS+13] of the API of the SUT.

- **Alphabet models** select, parameterize, and combine generic learning alphabet symbols in order to form a learn alphabet, fully instantiating/parameterizing the symbol models. They can easily be tuned according to a given risk profile via specific data-flows.
Test case models represent a membership query [SHM11] in the learn process, which are generated fully automatically during the model inference process.

The potential of this approach becomes particularly apparent under a risk-based perspective of system migration and functional evolution. Risk analysts are provided with a modeling level where they can build executable alphabet models that typically remain valid during the system lifecycle. The realization of these models is based on our hierarchical higher-order process modeling approach, supporting a simplicity-oriented version of higher-order process passing with full-code generation capabilities. In fact, based on the higher-order concepts, services and processes can be selected, modified, constructed and then safely passed as if they were data. Though unlike data they may be plugged into activities and executed (played) dynamically. This plug&play approach allows one to add new services, components, and processes without the need to change the system or interrupt the running processes. This way, we are able to generate alphabet models to executable code and, since the control flow is still accessible and mutable, the test case models may be created on the fly during learning. The benefit of this rigorous and highly automated model-driven approach is that it supports reuse, separation of concerns, and automation.

References


