Using DSLs for Software Testing

Mark Micallef[1] and Christian Colombo[2]

University of Malta

In her widely cited paper about the future of software testing, Bertolino [1] claims that domain specific languages (DSLs) have emerged as an efficient solution towards allowing experts within a domain to express specifications in that domain. She goes on the claim that success of domain-specific approaches should be built upon and extended to the testing stage of software engineering. An intuitive place to start would be to explore DSLs in the context of software testing such that languages constructed by domain experts can be leveraged to specify not only requirements but also test cases which validate those requirements. In this talk, we present and discuss the outcomes of three exploratory case studies which we carried out in order to investigate the utility of DSLs as applied to specifying tests in different domains, with each case study focusing on a particular aspect/characteristic of this application of DSLs. The three case studies are as follows:

Android Application Testing - In this case study, the focus was on investigating the possibility of designing a language which was able to express tests over the domain of applications developed for the Android platform. The main characteristic of such applications is that they exist in a domain, which is sufficiently different from other domains (e.g. desktop applications) to merit a domain-specific approach yet whose concepts are well-defined in official documentation [2] and understood by technical and non-technical stakeholders alike. In the study we developed a DSL which merged concepts from the Android platform with concepts from the domain of software testing thus allowing stakeholders to specify tests. Furthermore, we developed a prototype which implemented a subset of the features of the language as a proof-of-concept that the approach is feasible.

Graphical Games Testing - Whilst existing approaches to test automation sufficiently cater for the testing of "traditional" software systems which incorporate a standard set of user interface components, the same cannot be said when it comes to graphical games. Such games do not provide standard user interfaces and there is rarely any documentation which explicitly defines the domain. Hence in this case study we investigated the challenges involved in designing a DSL that expresses tests over a loosely-defined domain as well as the technical challenges involved in the execution of tests over graphical games. This work lead to the design of an extensible DSL for specifying tests over the domain of graphical games and the implementation of a prototype which executed tests against two popular games in the market.

Technology-Agnostic Test Automation for B2B Websites - Readers who purchase items online from a variety of online stores are likely familiar with notions such as a *product search*, a *shopping cart*, *checkout process*, and so on. These notions are basic building blocks which are found in most B2B websites. Yet when it comes to specifying and implementing automated tests in these domains, companies are forced to

start from scratch and build custom automation frameworks for their websites. In this case study we explored the idea of designing a DSL which is generic enough to express tests over B2B websites. We then used a classifier-based technique and prototype which demonstrates that it is possible for one to specify and automatically execute tests in the DSL without needing to be intimately familiar with the technical details of the website. This approach differs substantially from the current state of the art in which test engineers need to use application-specific hooks in order to implement automated tests. We believe this approach is likely to (1) encourage more focus on *what* the application should be doing rather than *how* it is doing it and (2) reduce the phenomenon of brittle tests whereby automated tests break as a system evolves, even if evolution is of a cosmetic nature.

Future work in this area will proceed down two tracks. Firstly, at least in the short term, we would like to maintain an exploratory mentality in which we carry out case studies across a number of domains in order to make observations and identify interesting research areas. Secondly, we would like to identify promising prototypes which arise from this work and continue to invest time in making them more feature-complete with a view of being able to apply them to real-life case studies with our industry collaborators.

References

1. A. Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, pages 85–103. IEEE Computer Society, 2007.
2. Google. Android user interface overview.