PhD Workshop

DRAFT Programme

18:00 – 18:20

T.J. Steenvoorden (Radboud University Nijmegen): *Type Directed Interactive Workflow Modelling.*

“Workflow management systems tend to be complicated, containing many basic elements to design and construct workflows. Also, coming from a workflow specification to an application usable by the end user is a lot of effort. Novel ways to generate this kind of applications, like the functional programming oriented iTasks system, are promising, but still too complicated to use for non-programmers. We present a graphical design tool aiding in specifying work and data flows in a straight forward and modular way, suitable for domain experts and workflow designers. Our tool discerns itself from others by incorporating the information needed to perform a task, thus combining workflows and dataflows into a single concept we call task flows. These flows have a visual representation inspired by Petri nets as well as a formal language representation inspired by natural language. Because all information in our workflows is typed, we are able to generate full blown web applications by using the iTasks system as a backend.”

18:20 – 18:40

T. Brunner (ELTE Budapest): *Code comprehension with CodeCompass*

“Bugfixing or new feature development requires a confident understanding of all details and consequences of the planned changes. For long existing large telecom systems, where the code base have been developed and maintained for decades by fluctuating teams, original intentions are lost, the documentation is untrustworthy or missing, the only reliable information is the code itself. Code comprehension of such large software systems is an essential, but usually very challenging task. As the method of comprehension is fundamentally different from writing new code, development tools are not performing well. 4 During the years, different programs have been developed with various complexity and feature set for code comprehension but none of them fulfilled our special requirements. CodeCompass is an open source LLVM/Clang based tool developed by Ericsson and the Eotvos Lorand University, Budapest, to help understanding large legacy software systems. Based on the LLVM/Clang compiler infrastructure, CodeCompass gives exact information on complex C/C++ language elements like overloading, inheritance, the (read or write) usage of variables, possible calls on function pointers and the virtual functions - features that various existing tools support only partially. The wide range of interactive visualizations extends further than the usual class and function call diagrams; architectural, component and interface diagrams are a few of the implemented graphs. To make comprehension more extensive, CodeCompass is not restricted to the source code. It also utilizes build information to explore the system architecture as well as version control information when available: git commit history and blame view are also visualized. Clang based static analysis results are also integrated to CodeCompass. Although the tool focuses mainly on C and C++, it also supports Java and Python languages. Having a web-based, pluginable, extensible architecture, the CodeCompass framework can be an open platform to further code comprehension,

static analysis and software metrics efforts. https://github.com/ericsson/codeCompass”

18:40 – 19:00

G. Horvath (ELTE Budapest): *Static Analysis with CodeChecker*

“Static analysis allows code optimisation, large-scale refactoring, code metrics and visualization in addition to discovering potential code defects. Static analysis involves the analysis of software without executing it, in almost all cases via an automated tool. This approach has its limitations, but it has been proven practical, and there are numerous issues and ideas leaving it an active field of research. The main idea behind static analysis' industrial application is cost-effectiveness. It has been statistically proven that the earlier an issue with the code is found, the easier, and cheaper it is to fix, while early triaging also enables easier long-term maintenance and code quality. For the C language family, the international open-source project LLVM/Clang
offers an easy to use, object-oriented syntax model, upon which the Clang Static Analyzer has been built. This tool allows the analysis of software code based on "checks", which are rulesets in the analyzer for finding call and data flows, control structures, and various other patterns that show potential defects in the code.”

19:00 – 19:20

Rui Pereira (Minho University Braga): *Energy Efficiency Across Programming Languages*

“This talk presents a study of the runtime, memory usage and energy consumption of twenty seven well-known software languages. We monitor the performance of such languages using ten different programming problems, expressed in each of the languages. Our results show interesting findings, such as how slower/faster languages consuming less/more energy, and how memory usage influences energy consumption. We also show how to use our results to provide software engineers support to decide which language to use when energy efficiency, time, or memory is a concern.”