**Motivation**

The growth of software complexity raises the challenges for maintaining its quality and keeping the presence of faults at minimum. The determination of the nature and location of a fault is tedious and time-consuming activity. Software fault localization is a research field helping developers in this effort with automated techniques. The thesis introduces a new such tool with focus on extensibility and user experience, overcoming weaknesses of existing applications.

**Problem Definition**

The task is to provide user with a helpful report of suspicious program elements to reduce the amount of time and effort committed to fault localization. We employ dynamic approach which augments the program with tracing calls to gather test-suite runtime data. Static analysis is performed as well. Generated data are used to estimate the suspiciousness of certain program elements (e.g., statements).

**Shortcomings of Existing Tools**

- Mostly research prototypes with poor integration into preferred workflow and toolchain (e.g., editor or programming language) of the user.
- Often make impractical assumptions. For example, they do not provide a rationale for the suspiciousness calculation or a context.
- Not publicly available or abandoned. To our best knowledge, only one available, actively developed “competitor”.

**Solution**

We present a highly modular design that enables adding support for preferred programming language, user interface or state-of-the-art localization technique, merely by implementing respective component. Internal APIs encourage plugins developers to provide the user with rationale, source code context and additional details. Data structures are rich-enough to support majority of techniques published in the literature and commonly-used programming languages. Initially implemented techniques employ probabilistic reasoning or inference of program invariants.

**Architecture**

The tool is composed of several components. Frontends process the source program. Core gathers the data and offers them via convenient API to plugins. The results are then delivered to the user through interface of choice.

**Contributions**

- Design of a new SFL tool that is extensible, language-agnostic and user-friendly. These goals are a significant improvement compared to existing solutions.
- Implementation of proposed framework, initially supporting three diverse localization techniques and two diverse programming languages (C and Python).
- The tool is openly available at https://github.com/aardwolf-sfl/aardwolf

**Results**

Best techniques detected up to 55% of bugs in their top 10 list on a standard dataset – a result comparable with the literature. The tool was successfully integrated with two bigger real-world projects (namely matplotlib and LibTIFF), and the process required only around 40 lines of configuration for each and no change in existing workflow. A sample of the localization output is depicted in figure 1.

**References**