SoftVarE – Thesis Topics of Paul Bittner

Evolution of Feature Traceability
About Me

My name is Paul

2014 – 2020: Study Computer Science in Braunschweig

Since then: PhD student at SP in Ulm

My research is about

... evolution of feature traceability,

... keeping forks and branches synchronised,

... benefit from product-line technology with less effort.

paul.bittner@uni-ulm.de
YOUR NEW EMPLOYEE
WHICH PART OF THE CODE
IS IMPLEMENTING THAT?
Feature Traceability Problem

Feature Traceability is the knowledge where each feature is implemented.
Software Product Lines

Feature-Modell

Feature-Auswahl

Generator

Wiederverwendbare Implementierungs-artefakte

Fertiges Program
**Problem**: Differencing of implementation artifacts (e.g., source code) is currently text-based, which does not respect syntax and is less accurate.

**Task**: Develop a variability-aware tree difference that operates on variation-aware abstract syntax trees (ASTs).

1. variability-aware parsing of old and new state using existing tools
2. use or extend existing tree differencers to work on variability-aware ASTs
3. ensure that the resulting diff is a valid (by construction)
Sending Feature Traces Back in Time

1. Some feature mapping is retrieved in the present.
   ```c
   void jump() {
     #ifdef GAMEPAD_SUPPORT
     gamepad.vibrate(0.1f);
     #endif
   }
   ```

2. Find last edit to mapped code (e.g., with git blame).

3. Introduce feature mapping in the past.
   ```c
   void jump() {
     #ifdef GAMEPAD_SUPPORT
     gamepad.vibrate(0.1f);
     #endif
   }
   ```

4. Run Feature Trace Recording to send feature mapping broadcast, accumulate, refine more knowledge

---

Feature location methods recover lost or unknown feature mappings in an existing code base manually, semi-automatically, or automatically. Feature location methods may use some existing feature mappings as input. In case few or no feature mappings are known, can we assist tools with feature mappings from the future?

**Task:** Develop and implement a method for sending feature mappings into the past.

**Questions:**
- How beneficial is sending feature mappings back in time?
- Can we compute further feature mappings by sending them through time?
- How far can we send feature mappings back in time (at most, on average, at least)?
- Does traveling back to the future yield more or more accurate feature mappings?
Sending Feature Traces into the Future with Feature Trace Recording

void pop() {
    storage[head--] = null;
}

(1) insert
context: SafeStack

Stack<T> pop() {
    Stack<T> c = clone();
    if (!empty()) {
        c.storage[c.head--] = null;
    }
    return c;
}

(2) update
context: ImmutableStack

void pop() {
    if (!empty()) {
        storage[head--] = null;
    }
}

(3) move
context: null

void pop() {
    if (!empty()) {
        storage[head--] = null;
    }
}

(4) delete
context: ImmutableStack

void pop() {
    if (!empty()) {
    }
}

(5) insert
context: ImmutableStack
Sending Feature Traces into the Future
with Feature Trace Recording

Feature trace recording is a feature location method that monitors source code evolution. Feature traces are recorded while developers are programming (instead of recovering feature traces retroactively as variability mining does). For example, in the image above you see how edits are made to an initial version of the pop method of a Stack in Java. Upon each edit, developers specify which feature they are editing (which we call the feature context). Feature mappings, indicated by coloured source code) are then recorded upon the edit. We thus know to which feature the edited code belongs to.

Currently, feature trace recording was not evaluated on large commit history history so its benefits are mostly unexplored. In particular its capability of keeping feature mappings consistent across evolution as well as gathering further feature mappings over the course of evolution could be promising to gradually increase developers’ knowledge on feature traceability. Moreover, feature trace recording is subject to various parameters such as quality of the feature context, how often a feature context is specified, or the granularity of the observed edits.

Tasks:
• Implement feature trace recording into our VariantSync framework.
• Generate variants and edit histories from software product lines (for example Linux).
• Run feature trace recording on the generated histories.
• Evaluate quality of the recorded feature traces.