SoftVarE – Thesis Topics of Paul Bittner
About Me

• My name is Paul
• 2014 – 2020: Study Computer Science in Braunschweig
• Since then: PhD student at Institute of Software Engineering and Programming Languages in Ulm
• My research is about helping people to ...
  • ... find their code
  • ... keep their forks and branches synchronised
  • ... benefit from SPL technology with less effort

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

Feature Traceability is the knowledge where each feature is implemented.
Traceability is given in *software product lines* ...  

... but in practice variability is often implemented by copying (or branching) and adapting existing code.

So how can we help developers to *document* and *maintain* feature traces here?
Clone-and-Own as Prominent Variability Approach
Clone-and-Own as Prominent Variability Approach

Variant 1

Variant 2

Variant 3

Duplicate Code

? ?

git branch

Bug

 WHICH PART OF THE CODE IS IMPLEMENTING THAT?
Topics on Feature Traceability
void foo() {
    int x = pi();
    /* ... */
}

int pi() {
    return 3;
}

#ifdef MathUtils
int pi() {
    return 3;
}
#endif
Type-Checking Feature Traces in Future Variants

Feature traces identify the implementation of features in the source code of a software system. *Feature location*, is the process of recovering lost information on feature traces. One possible method to recover feature traces is *variability mining* [1] where dependencies between types in the used programming language are inspected (semi-)automatically. For example, if we know that a function implements a feature $A$, we know that all calls to that function also have to trace to $A$, or a formula stronger than $A$ (i.e., a function call has to be mapped to a feature formula $X$ that satisfies the constraint $X \Rightarrow A$). Thus, we ensure that whenever we call that function in a software variant, that function is also present. However, some dependencies might not be present in the edited variant but other ones. The goal of this thesis is to extend this type checking to consider other variants as well.

**Tasks:**
- Extend and implement variability mining in our VariantSync prototype IDE.
- Evaluate your results on existing software projects as in previous work on Variability Mining.
Feature-Oriented Commits

1. `void pop() { storage[head--] = null; }`
   - Context: SafeStack
   - Change: Insert

2. `void pop() { if (!empty()) { } storage[head--] = null; }
   - Context: null
   - Change: Move

3. `void pop() { if (!empty()) { storage[head--] = null; } }
   - Context: ImmutableStack
   - Change: Insert

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

Today's standard for tracking source code evolution are version control systems such as Git and Subversion. So histories in version control systems are the most abundant datasets for evaluating and employing feature trace recording. However, although intended in version control, commits are rarely disciplined in the sense that each commit captures exactly one change to a feature. Sometimes, even huge refactorings or renamings affecting the entire code base occur in a single commit.

Related Work: [2], [3]

Tasks:
- Literature survey
- Find a way to detect if a commit is disciplined
- Investigate if and how undisciplined commits can be divided into smaller disciplined edits.
- Evaluate or prove that your division maintains consistency.
- Evaluate the usefulness of your method: How often are commits undisciplined? How many undisciplined commits can be made disciplined?
Topics on Diffing and Merging
Semantic Lifting of Abstract Syntax Tree Edits

foo(bar);

if (baz) {
  foo(bar);
}
Semantic Lifting of Abstract Syntax Tree Edits

Existing tree differencers yield:
Technical Low-Level Diff

What we need:
Semantic User-Level Diff
Semantic Lifting of Abstract Syntax Tree Edits

Questions:

- Can we lift edit scripts to semantic-edit scripts?
- Do semantic edits represent developers actions more accurately?
- Do existing tree diffing algorithms yield edits impossible for programmers to execute on the concrete syntax?
- Are semantic edits sufficient to assess developers' intents?
- Does semantic lifting increase diffing accuracy?
- Can we use state-based diff (based on set operations) to lift edit scripts?

Related Work: [4-7]

Goal: Create an algorithm for semantic lifting of edit scripts obtained from off-the-shelf tree differs to semantic edits. Optionally, extend the notion of semantic edits for increased accuracy. Evaluate your results.
References


(2) P. M. Bittner, A. Schultheiß, T. Thüm, T. Kehrer, J. M. Young, L. Linsbauer, “Feature Trace Recording”, ESEC/FSE 2021


(7) P. M. Bittner, “Semi-Automated Inference of Feature Traceability During Software Development”, Master’s Thesis