Code Clones in Software Product Lines

PASED Summer School 2011, Montreal

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG — INF

Code Clones

Code Clones = code fragments replicated across the source code

Negative effects on software system:

- increased effort for maintenance
- error propagation
- inconsistent changes

Root causes for code clones:

- reuse by *Copy&Paste(&Adaption)*
- missing abstraction mechanisms (language-dependent)
- insufficient knowledge of source code

```
class Graph { /*...*/
  public Graph Kruskal {
    // some code
    for(int j=0;j<(vrep.members).size();j++) {
      vaux = (Vertex) (vrep.members).get(j);
      vaux.representative = urep;
      (urep.members).add(vaux);
    }
    /*...*/
    for(int j=0;j<(urep.members).size();j++) {
      vaux = (Vertex) (urep.members).get(j);
      vaux.representative = vrep;
      (vrep.members).add(vaux);
    }
  }}
```

} Clone A

} Clone B

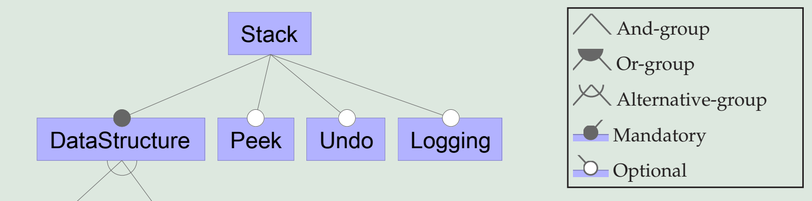
Research Questions

A. Do code clones exist in (compositional) SPLs ?
B. What are causes for code clones in software product lines ?
C. To what extent are they caused by specific SPL mechanisms ?
D. What are relations between features containing code clones?

Software Product Lines

Software Product Line (SPL) = set of software systems, that share a common, managed set of features

- can be used to introduce and manage variability in SW systems
- *Feature* = increment in end-user-visible functionality
- selecting a subset of features to generate a tailored program
- valid feature combinations are specified using a *feature model*



Two approaches:

1. *Annotative*
   - variability through source code annotation, e.g., preprocessor directives
   - *virtual* separation

```
class Stack {
  #ifdef Undo
  int backup;
  void undo() {/*...*/}
  #endif
  #ifdef Peek
  int peek() {/*...*/}
  #endif
  void push(int v) {
    #ifdef Undo
    backup=peek();
    #endif
    /*...*/
  }
  int pop() {/*...*/}
```

2. *Compositional*
   - features are implemented as cohesive units, e.g., *feature modules*
   - implemented using *feature-oriented programming (FOP)*
   - *physical* separation

Feature Stack

```
class Stack { ...
  void push(int v)
       {/*...*/}
  int pop() {/*...*/}
}
```

Feature Peek

```
refines class Stack {
  int peek() {/*...*/}
}
```

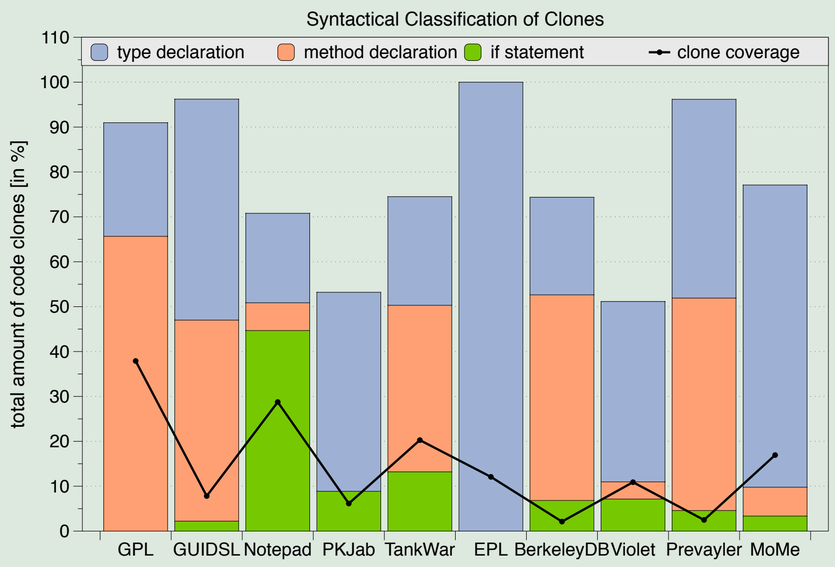
Feature Undo

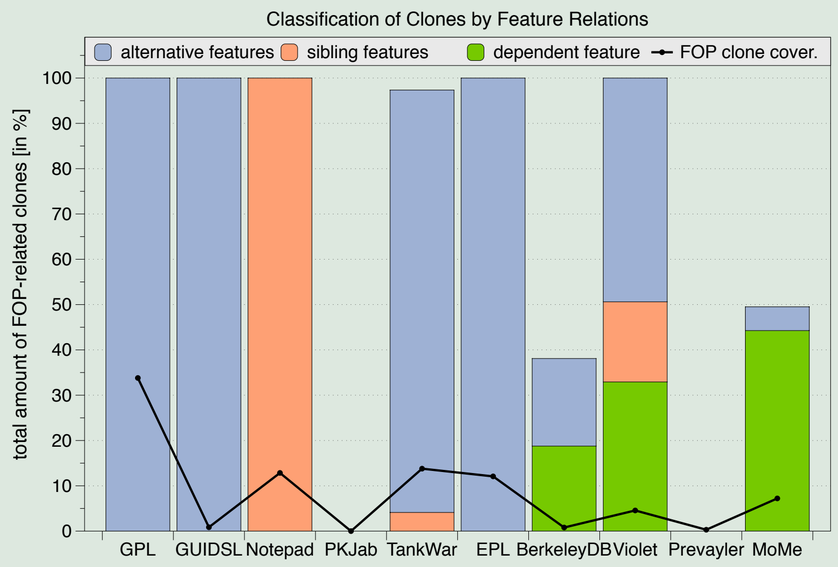
```
refines class Stack {
      ...
  int backup;
  void undo() {/*...*/}
  void push(int v) {
    backup=peek();
    original(v);
  }
}
```

Current Results

1. Code clones in *compositional* SPLs (answers for questions A, D, and partly C)

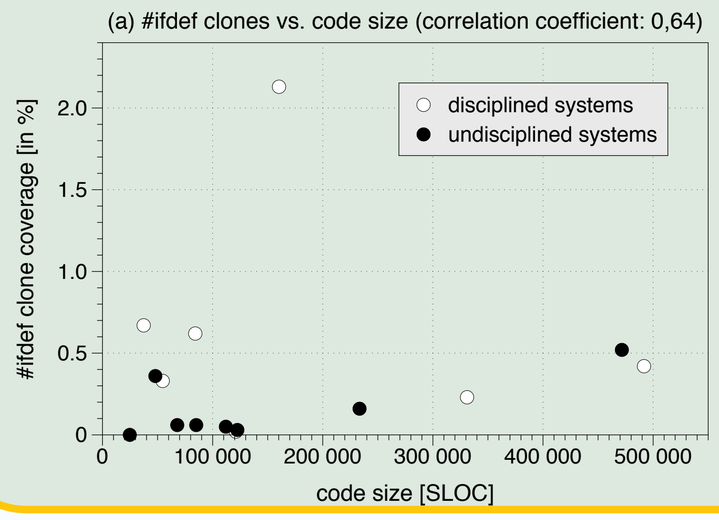
- case study on 10 Java systems (six developed from scratch, four refactored from legacy applications)
- code clones exist in these SPLs ⇒ to a considerable extent they are *FOP-related*, i.e., clones occur in at least two different features
- clones mainly occur between *alternative* features ⇒ good removal capabilities (using refactoring)

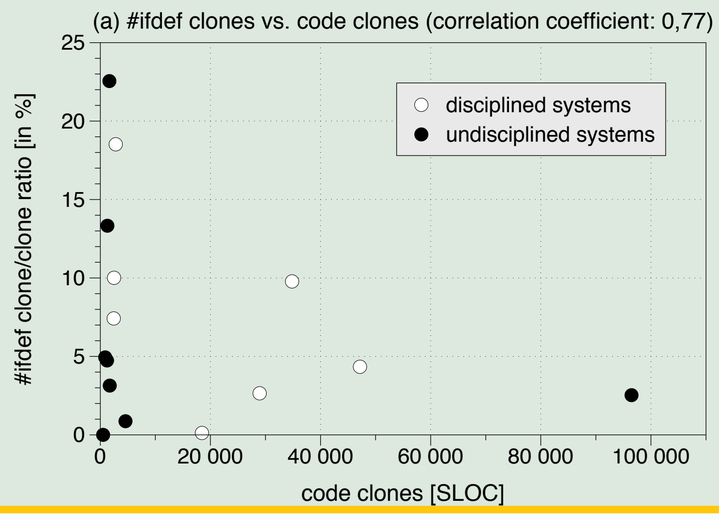


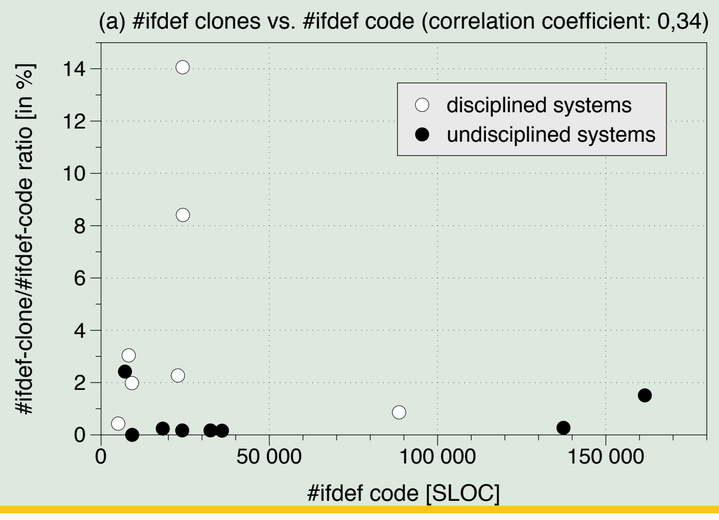


2. Effect of preprocessor annotations on code clones (answers for questions A, D, and B)

- case study on 15 systems written in C (25 to 500 KLOC)
- only a minor fraction of code clones occur in preprocessor annotations (called *#ifdef clones*)
- minor amount of #ifdef clones in systems with undisciplined annotations ⇒ disciplined annotations are prone to code clones
- next step: code clones vs. undisciplined annotations







Future Work

- detailed analysis of code clones in annotative SPLs
- annotative vs. compositional SPLs (w.r.t. clones)
- product-line aware refactoring for code clone removal
- analyzing code clones in the evolution of SPLs

References

[1] S. Schulze, S. Apel, C. Kästner. Code Clones in Feature-Oriented Software Product Lines. In *GPCE 2010*

[2] S. Schulze, E. Jürgens, J. Feigenspan. Analyzing the Effect of Preprocessor Annotations on Code Clones. In *SCAM 2011*

Contact

Sandro Schulze
School of Computer Science
University of Magdeburg
E-mail: sanschul@ovgu.de