

Visualizing Software Product Line Variability in Source Code



Christian Kästner
University of Magdeburg
Magdeburg, Germany



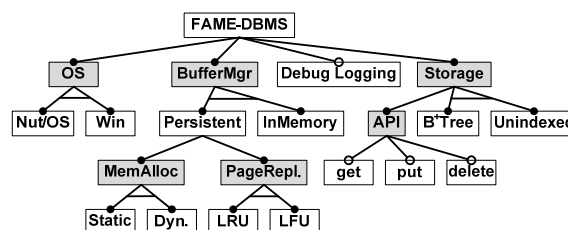
Salvador Trujillo
IKERLAN Research Center
Mondragon, Spain



Sven Apel
University of Passau
Passau, Germany

Software Product Lines (SPL) & Features

- Set of related software products for one domain
- Generated from common code base
- Products distinguished in terms of features
- Feature: domain abstraction relevant to stakeholders; increment in functionality



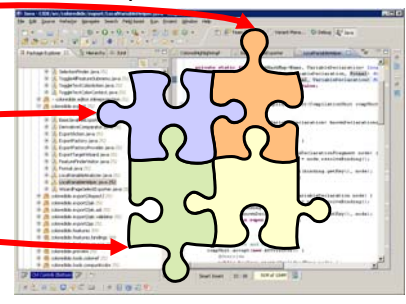
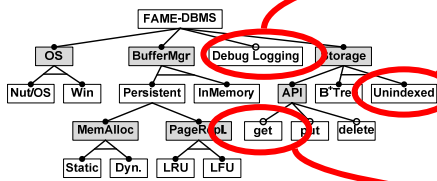
Challenge: Feature Traceability

Problem Space

Solution Space

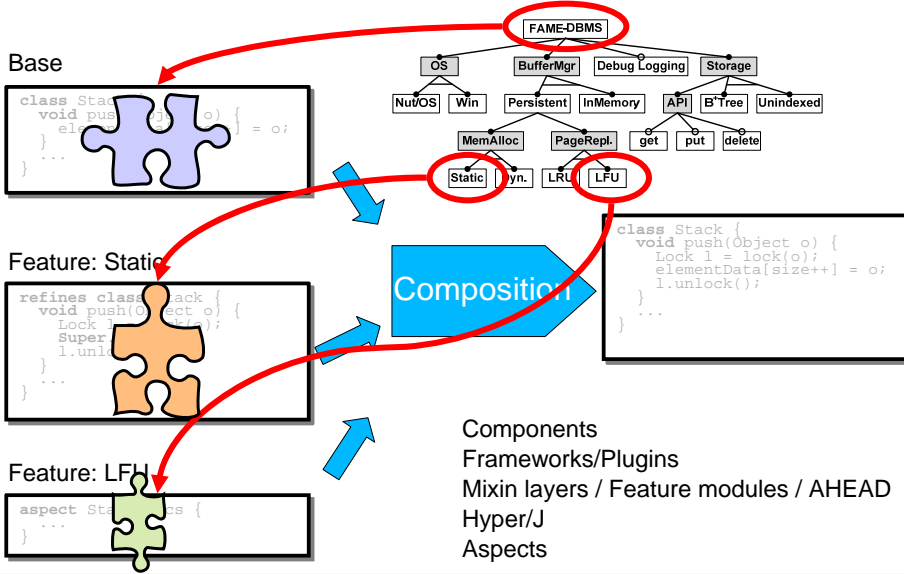
(Feature Model)

(Implementation)



There is a bug in B*Tree changes:
How to find and understand it if code is flat code?

Academia: Compositional Approaches



Pros and Cons of Compositional Approaches

- Good Modularity
- Good Traceability

- Only coarse granularity
- Might cause architectural overhead

- Hard to adopt
 - ◆ Breaks tool chain
 - ◆ Force new paradigm upon existing project



Industry: Annotative Approaches

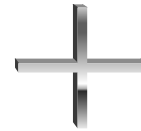
```

class A {
class Lock {
class LFU {
class Base {
class Stack {
void push(Object o) {
#ifdef WIN
Lock l = lock(o);
#endif
elementData[size++] = o;
#ifdef WIN
l.unlock();
#endif
#ifdef DYNAMIC
#endif
}
            
```

Preprocessors
Frames/XVCL
Gears
pure::variants

Pros and Cons of Annotative Approaches

- Easy to adopt
- Fine granularity
- No overhead
- Loss of Modularity
- **Poor Traceability**



**Focus of this talk: (intent of this work)
Achieve traceability for annotative approaches**

Virtual Separations of Concerns with CIDE

- Annotations with background colors (similar `#ifdef`)
- Managed by tool infrastructure
- Allows novel concepts for views and navigation

```

class Stack {
    public Stack(int maxSize, PrintStream loggingTarget) {
        elementData = new Object[maxSize];
        logTarget = loggingTarget;
    }

    private int size = 0;
    private Object[] elementData;
    private PrintStream logTarget;

    void push(Object o, Transaction tsm) {
        if (o == null || tsm == null)
            return;
        synchronized (lock()) {
            elementData[size++] = o;
            logTarget.println("push: " + o + ", new size: " + size);
            fireStackChanged();
        }
    }

    public Object pop(Transaction tsm) {
        ...
    }
}

```

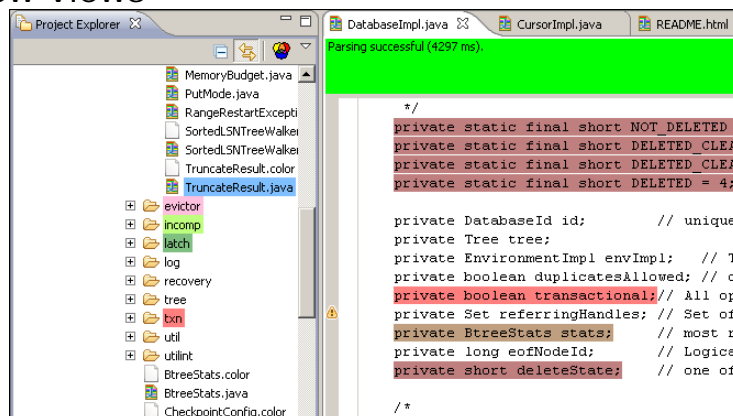
Discussion: Use of Colors

- Colors are intuitive to map
- Code not obfuscated
- Background color not used in most editors
- Humans can only distinguish few colors clearly
- Colorblindness is common
- Colors mapped to features 1:n (only 12 colors)
- Colors indicate start and end of feature code
- Features looked up or inferred from context

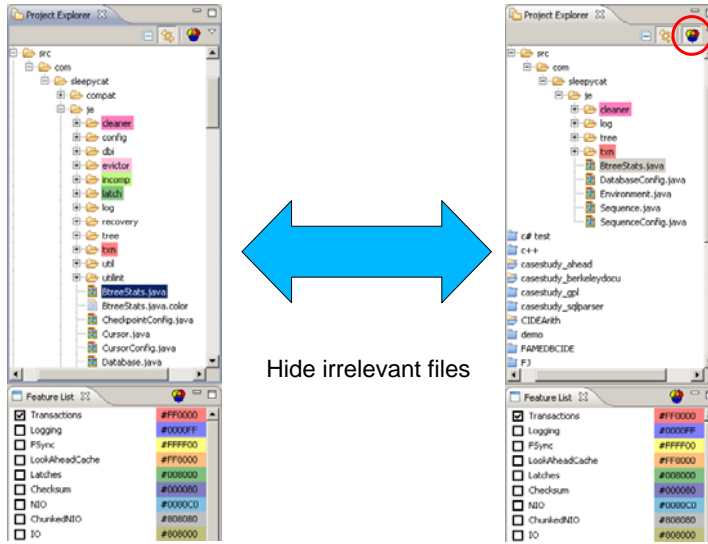


Scaling CIDE (what is new)

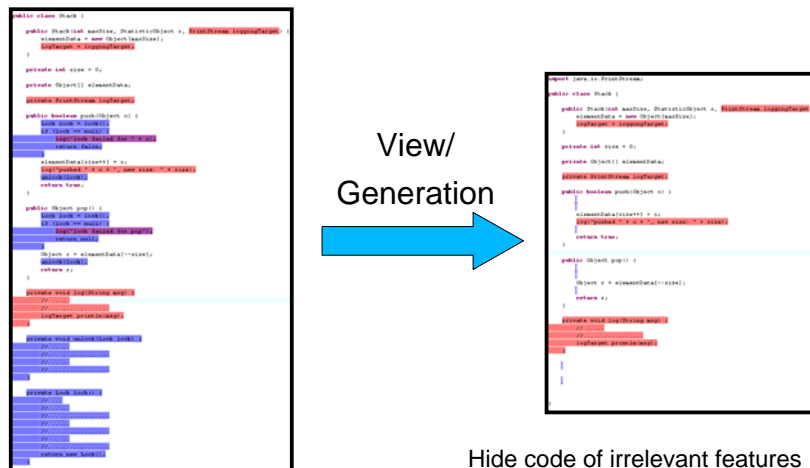
- Annotate code fragments (select+assign)
- Annotate entire directories and files (NEW!)
- New views



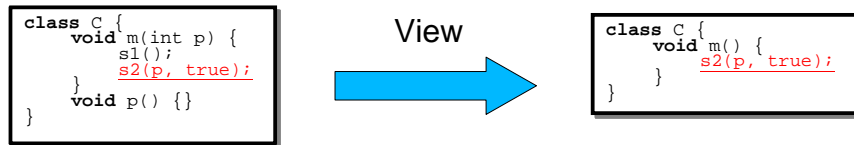
View on File System



Views on File Content (Variant View)



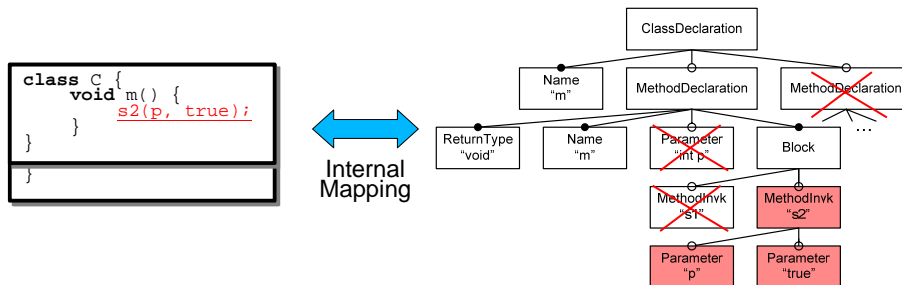
Views on File Content (Realization View)



Show only feature's code and context

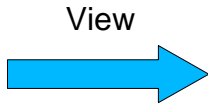
Underlying Structure

- CIDE uses the underlying AST
- Features assigned to optional AST-subtrees
- AST hidden from developer, mapped by tool



Realization View – Another Example

```
<project name='Release G.'>
<description> ... </description>
<patternset id='tool.patterns'>
  <include name='modeexplorer**'/>
  <include name='ant**/*.jar'/>
  <include name='applybali2jak**'/>
  <include name='xak**'/>
  <include name='infozone**'/>
  <include name='saxon**'/>
</patternset>
<target name='ahead' ...>
  <antcall inheritall='false' ...>
    <param name='dir.source' ... />
  </antcall>
</target>
</project>
```

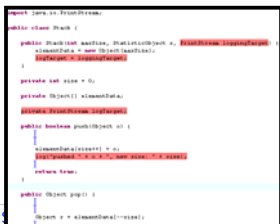
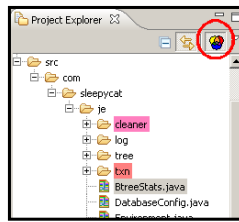
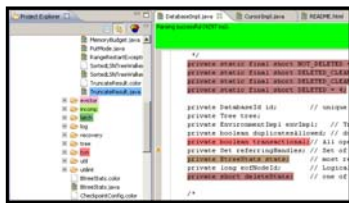


```
<project>
  <patternset>
    <include name='xak**'/>
  </patternset>
</project>
```

CIDE handles source code, XML, ..

Traceability Achieved

- Annotate code base
- View on file system
- Variant view
- Realization view



```
class C {
  void m() {
    s2(p, true);
  }
}
```

Related Work

- FEAT, AspectBrowser, JQuery, Spotlight
 - ◆ Explore or query legacy applications
 - ◆ No incentive to keep model up to date
- Mylyn
 - ◆ Context aware view on file system
- Program slicing
 - ◆ View on control flow for debugging
- FeatureMapper
 - ◆ "CIDE for models"



Conclusion

- Traceability important for SPL maintenance
- Situation
 - ◆ Direct traceability in compositional approaches (academia);
 - ◆ not in annotative approaches (industry)
- Traceability achieved with CIDE
 - ◆ Use existing feature annotations
 - ◆ See only relevant files
 - ◆ See only relevant parts of files