Photo (cc) Horia Varlan

Virtual Separation of Concerns

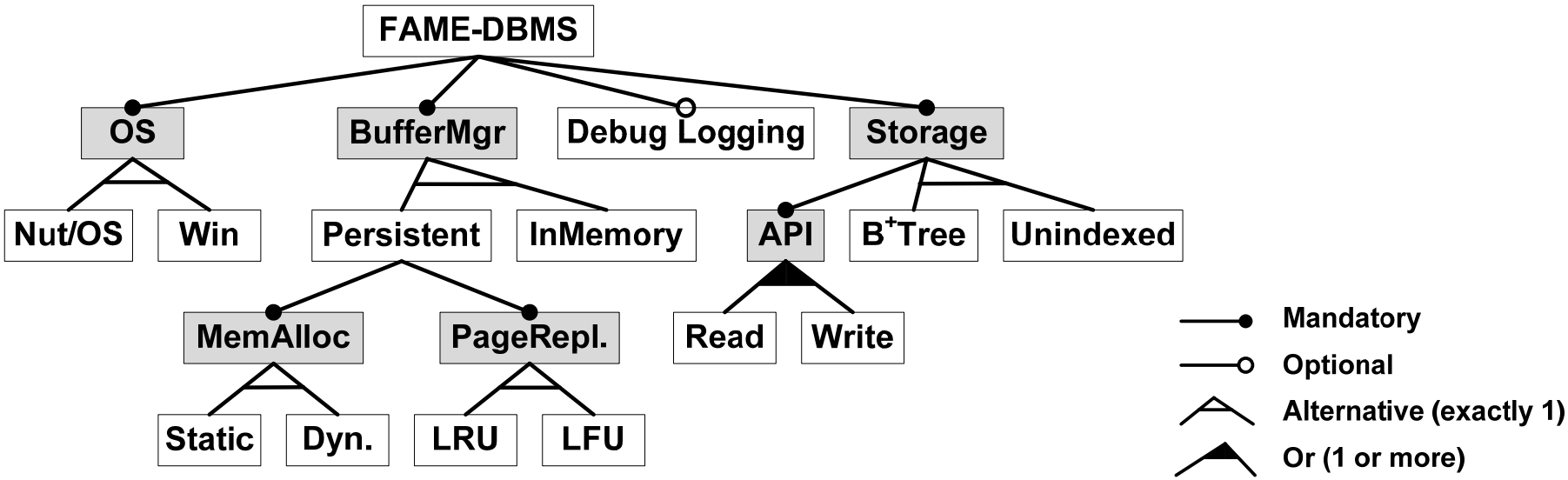
Toward preprocessors 2.0

Christian Kästner

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Software Product Lines & Features

- Set of related software systems (variants) in a domain
- Generated from **common code base**
- Variants distinguished in terms of **features**



Conditional Compilation with Preprocessors

- Annotated code fragments
- Variants with and without features
- Common for product-line implementation in **industry**

```
static int _rep_queue_filedone(...
    DB_ENV *dbenv;
    REP *rep;
    __rep_fileinfo_args *rfp; {
#ifndef HAVE_QUEUE
    COMPQUIET(rep, NULL);
    COMPQUIET(rfp, NULL);
    return (__db_no_queue_am(dbenv
#else
    db_pgno_t first, last;
    u_int32_t flags;
    int empty, ret, t_ret;
#ifdef DIAGNOSTIC
    DB_MSGBUF mb;
#endif
    ...
```

Excerpt from Oracle's Berkeley DB

Preprocessors in C, C++, Java ME, Pascal, Erlang, C#, Visual Basic, ...
GPP, GNU M4, SPLET, Frames/XVCL, Gears, pure::variants

Objections / Criticism

Designed in the 70th and hardly evolved since

"#ifdef considered harmful"

"#ifdef hell"

"maintenance becomes a 'hit or miss' process"

"is difficult to determine if the code being viewed is actually compiled into the system"

"incomprehensible source texts"

"programming errors are easy to make and difficult to detect"

"CPP makes maintenance difficult"

"source code rapidly becomes a maze"

"preprocessor diagnostics are poor"

Academia: Compositional Approaches

Base / Platform

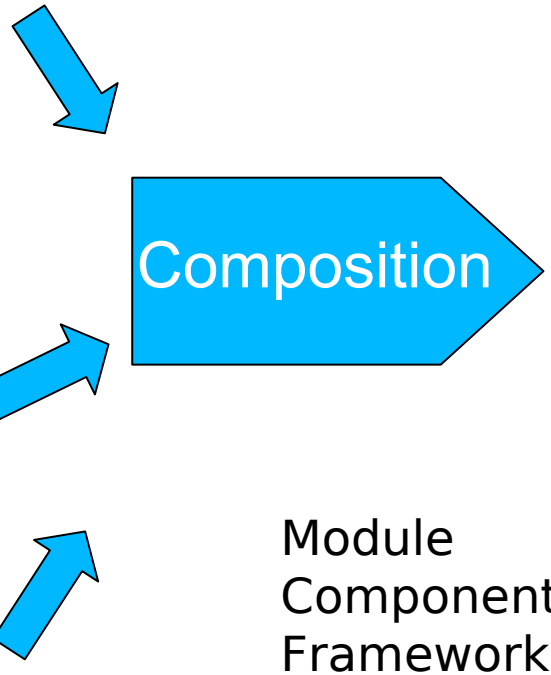
```
class Stack {
  void push(Object o) {
    elementData[size++] = o;
  }
  ...
}
```

Feature: Queue

```
refines class Stack {
  void push(Object o) {
    Lock l = lock(o);
    Super.push(o);
    l.unlock();
  }
  ...
}
```

Feature: Diagnostic

```
aspect Diagnostics {
  ...
}
```




```
class Stack {
  void push(Object o) {
    Lock l = lock(o);
    elementData[size++] = o;
    l.unlock();
  }
  ...
}
```

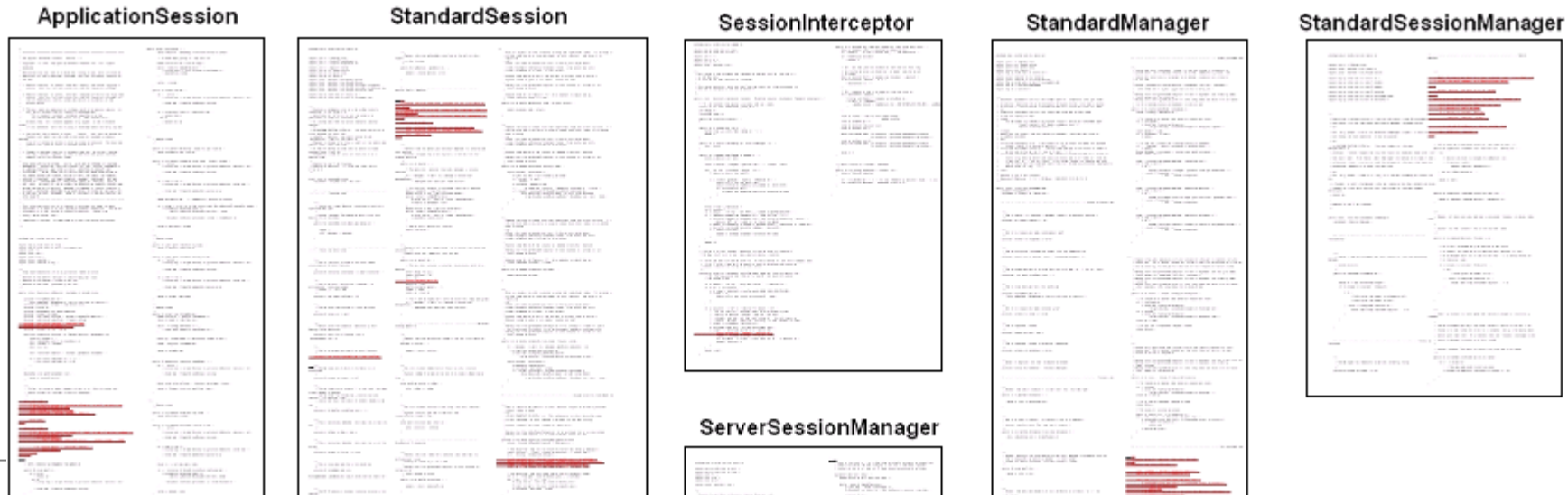
Module
Components
Frameworks, Plug-ins
Feature-Modules / Mixin Layers / ...
Aspects / Subjects, Hyper/J

Agenda

- Problems and Advantages of Preprocessors
- 4 Improvements
  - Views
  - Visual Representation
  - Disciplined Annotations
  - Product-Line-Aware Type System
- Summary and Perspective

Problem 1: Lack of Separation of Concerns

- Scattered and tangled source code
- Global search to understand a feature
- Deleting obsolete feature as challenge
- Difficult distributed development
- Difficult reuse

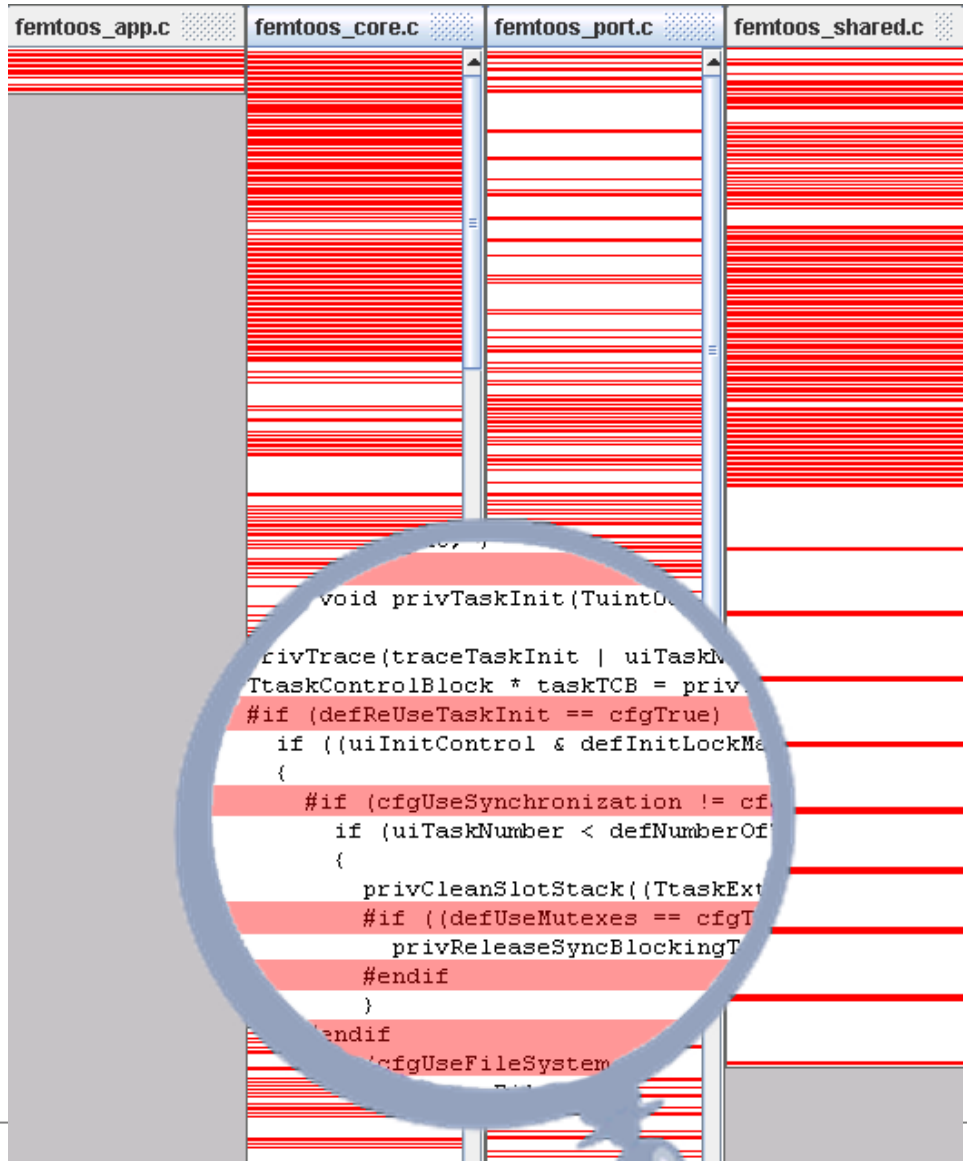


Problem 2: Obfuscation

- Mixture of two languages (C und #ifdef, ...)
- Control flow hard to follow
- Additional line breaks destroy code layout
- Long annotations difficult to recognize

```
class Stack {
    void push(Object o
#ifdef SYNC
    , Transaction txn
#endif
    ) {
        if (o==null
#ifdef SYNC
            || txn==null
#endif
            ) return;
#ifdef SYNC
        Lock l=txn.lock(o);
#endif
        elementData[size++] = o;
#ifdef SYNC
        l.unlock();
#endif
        fireStackChanged();
    }
}
```

Preprocessor in Femto OS



Problem 3: Error-Prone

- Syntax Errors

```
static int _rep_queue_filedone(...)
    DB_ENV *dbenv;
    REP *rep;
    __rep_fileinfo_args *rfp; {
#ifndef HAVE_QUEUE
    COMPQUIET(rep, NULL);
    COMPQUIET(rfp, NULL);
    return (__db_no_queue_am(dbenv));
#else
    db_pgno_t first, last;
    u_int32_t flags;
    int empty, ret, t_ret;
#ifdef DIAGNOSTIC
    DB_MSGBUF mb;
#endif
    // over 100 lines of additional code
}
#endif
```

- Type Errors

```
#ifdef TABLES
class Table {
  void insert(Object data,
              Txn txn) {
      storage.set(data,
              txn.getLock());
  }
}
#endif
class Storage {
#ifdef WRITE
   boolean set(…) { ... }
#endif
}
```

Problem 3: Error-Prone

- Syntax Errors

```
static int _rep_queue_filedone(...)
    DB_ENV *dbenv;
    REP *rep;
    __rep_fileinfo_args *rfp; {
#ifndef HAVE_QUEUE
    COMPQUIET(rep, NULL);
    COMPQUIET(rfp, NULL);
    return (__db_no_queue_am(dbenv));
#else
    db_pgno_t first, last;
    u_int32_t flags;
    int empty, ret, t_ret;
#ifdef DIAGNOSTIC
    DB_MSGBUF mb;
#endif
    // over 100 lines of additional code
}
#endif
```

- Type Errors

```
#ifdef TABLES
class Table {
  void insert(Object data,
              Txn txn) {
      storage.set(data,
              txn.getLock());
  }
}
#endif
class Storage {
#ifdef WRITE
   boolean set(…) { ... }
#endif
}
```

Problem 3: Error-Prone

- Syntax Errors

```
static int _rep_queue_filedone(...)
    DB_ENV *dbenv;
    REP *rep;
    __rep_fileinfo_args *rfp; {
#ifndef HAVE_QUEUE
    COMPQUIET(rep, NULL);
    COMPQUIET(rfp, NULL);
    return (__db_no_queue_am(dbenv));
#else
    db_pgno_t first, last;
    u_int32_t flags;
    int empty, ret, t_ret;
#ifdef DIAGNOSTIC
    DB_MSGBUF mb;
#endif
    // over 100 lines of additional code
}
#endif
```

- Type Errors

```
#ifdef TABLES
class Table {
  void insert(Object data,
              Txn txn) {
      storage.set(data,
              txn.getLock());
  }
}
#endif
class Storage {
#ifdef WRITE
   boolean set(…) { ... }
#endif
}
```

Advantages

- Easy to use
  - “annotate and remove”
  - Already part of many languages / simple tools
  - Most developers already familiar
- Flexible / Expressive
- Language-independent / Uniform
- Low Adoption Barrier
  - esp. with legacy code

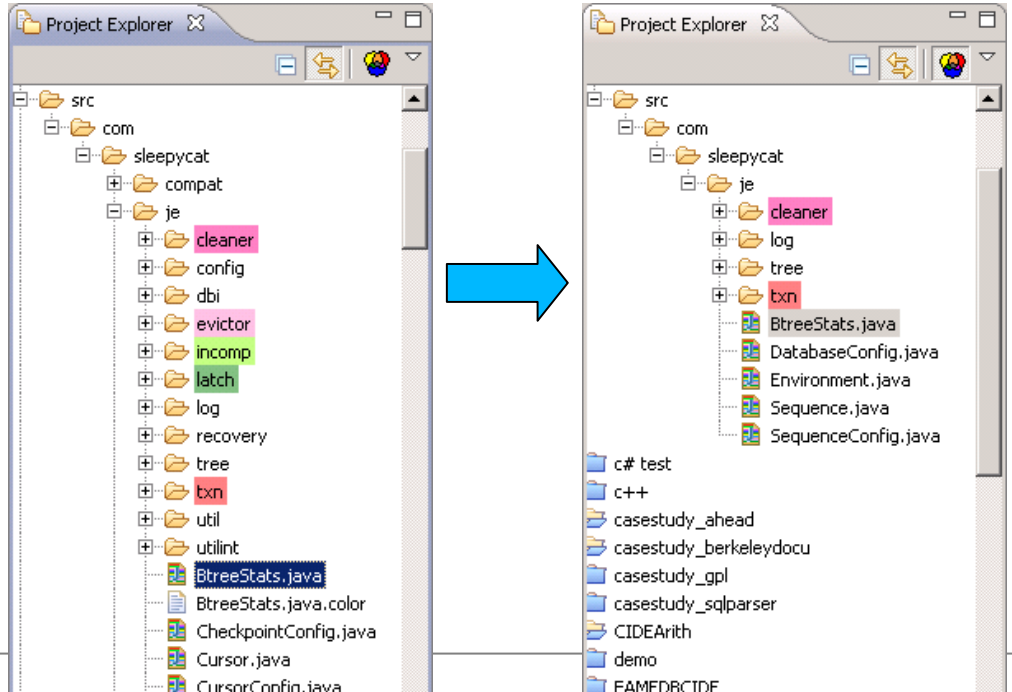
Agenda

- Problems and Advantages of Preprocessors
- 4 Improvements
  - Views
  - Visual Representation
  - Disciplined Annotations
  - Product-Line-Aware Type System
- Summary and Perspective

[ICSE'08, ViSPLE'08]

Views

- Emulate modularity
- Hide irrelevant source code
  - Hides entire files
  - Hides code inside files



Related work:
- C-CLR
- Version Editor
- FeatureMapper
- pure::variants
- Effective Views
- on-demand remodularization

...

Expression Problem

```
1  #ifdef ADD
2  class Add extends Expr {
3    Expr left, right;
4    Add(Expr l, Expr r)
5        {left=l; right=r;}
6  #ifdef EVAL
7    double eval() {
8      return left.eval() +
9          right.eval();
10   }
11 #endif
12 #ifdef PRINT
13   void print() {
14     left.print();
15     System.out.print("+");
16     right.print();
17   }
18 #endif
19 }
20 #endif
```

```
21 #ifdef POWER
22 class Pow extends Expr {
23   Expr base, exp;
24   Pow(Expr b, Expr e)
25       { base=b; exp=e; }
26 #ifdef EVAL
27   int eval() {
28     return MathLib.pow(
29         base.eval(),
30         right.eval());
31   }
32 #endif
33 #ifdef PRINT
34   void print() {
35     left.print();
36     System.out.print("^");
37     right.print();
38   }
39 #endif
40 }
41 #endif
```

```
42 class MathLib {
43   static int pow
44     (int b, int e)
45   {
46     if (e<=0)
47       return 1;
48     return b *
49       pow(b, e-1);
50   }
51   //...
52 }
```

View on Feature Eval

```
1  #ifdef ADD
2  class Add [...] {
3    [...]
4  #ifdef EVAL
5    int eval() {
6      return left.eval() +
7          right.eval();
8    }
9  #endif
10   [...]
11 }
12 #endif
```

```
13 #ifdef POWER
14 class Pow [...] {
15   [...]
16 #ifdef EVAL
17   int eval() {
18     return MathLib.pow(
19         base.eval(),
20         right.eval());
21   }
22 #endif
23   [...]
24 }
25 #endif
```

- Shows all files containing *Eval* code
- Shows context information (kursiv)
- Hides code without annotations

View on the Variant “Add and Eval”

```
1  #ifdef ADD
2  class Add extends Expr {
3    Expr left, right;
4    Add(Expr l, Expr r)
5        {left=l; right=r;}
6  #ifdef EVAL
7    int eval() {
8      return left.eval() +
9          right.eval();
10   }
11 #endif
12   [...]
13 }
14 #endif
```

```
15 class MathLib {
16   static int pow
17     (int b, int e)
18   {
19     if (e<=0)
20       return 1;
21     return b *
22       pow(b, e-1);
23   }
24   //...
25 }
```

- Entirely hides file *Power*
- Hides method *print* in *Add* ([...])
- Code without annotation remains visible

Agenda

- Problems and Advantages of Preprocessors
- 4 Improvements
  - Views
  - Visual Representation
  - Disciplined Annotations
  - Product-Line-Aware Type System
- Summary and Perspective

[ICSE'08, ViSPLE'08]

Visual Representation: Background Colors

```
class Stack {
    void push(Object o
#ifdef SYNC
    , Transaction txn
#endif
    ) {
        if (o==null
#ifdef SYNC
            || txn==null
#endif
            ) return;
#ifdef SYNC
        Lock l=txn.lock(o);
#endif
        elementData[size++] = o;
#ifdef SYNC
        l.unlock();
#endif
        fireStackChanged();
    }
}
```

```
1 class Stack {
2   void push(Object o, Transaction txn) {
3     if (o==null || txn==null) return;
4     Lock l=txn.lock(o);
5     elementData[size++] = o;
6     l.unlock();
7     fireStackChanged();
8   }
9 }
```

Features: SYNCHRONIZATION

Related Work:
- Spotlight
- NetBeans
- fmp2rsm
- FeatureMapper
- AspectBrowser

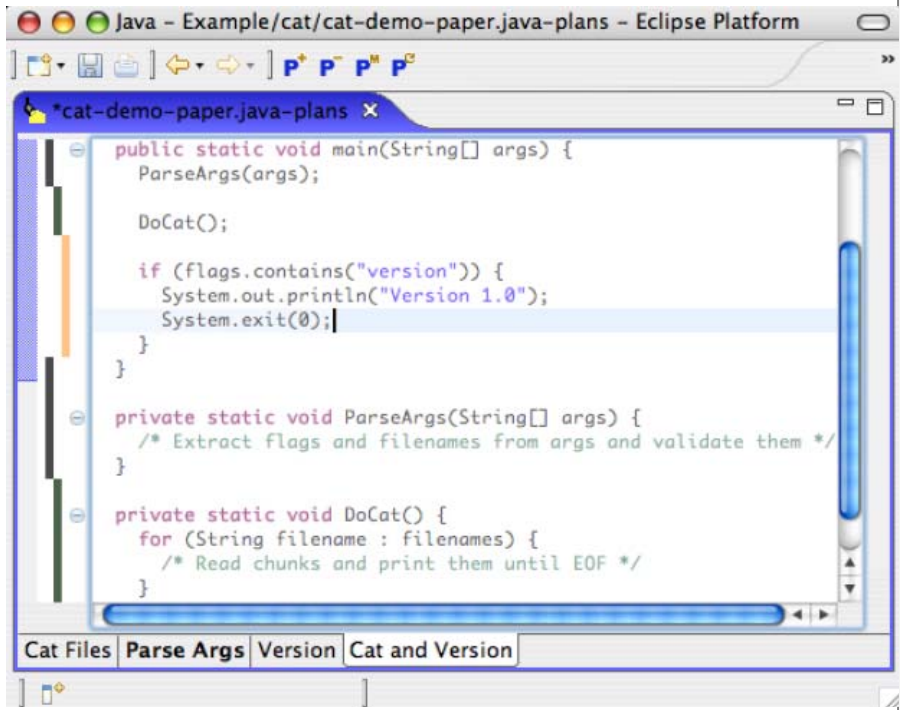
...

Alternative Visual Representations

- NetBeans

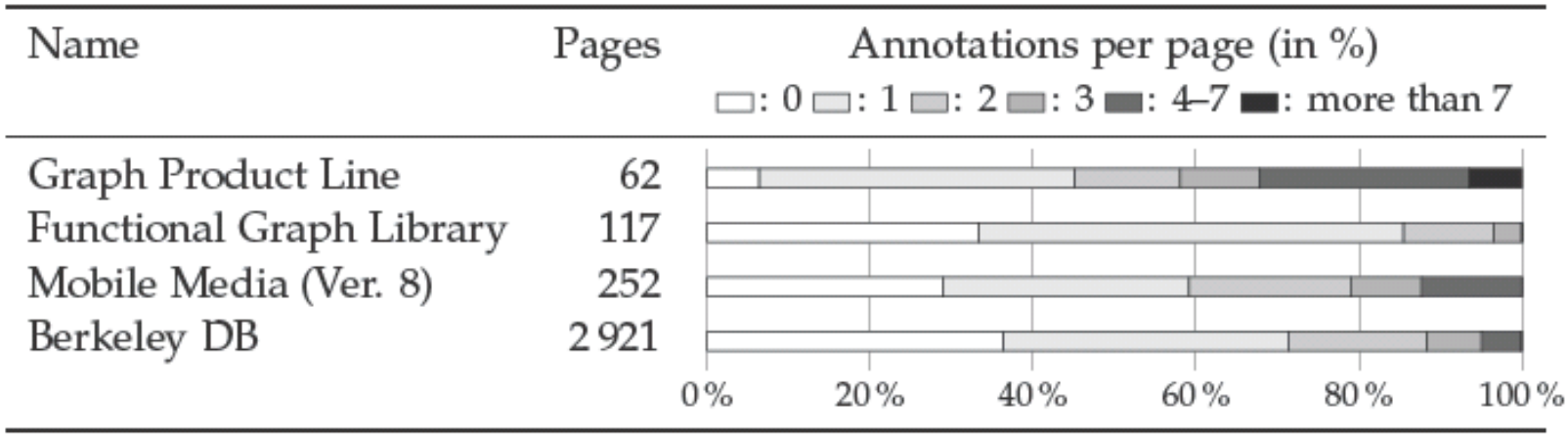
```
        /** Read HTML and if it has links, redirect and pars
        protected String parseHTMLRedirect(String url, Input
    throws IOException, Exception {
                //#ifdef DSMALLMEM
//#             throw new IOException("Error HTML not suppor
                //#else
                if (m_redirect) {
                        //#ifdef DLOGGING
//#                     logger.severe("Error 2nd redirect ur
                        //#endif
                        System.out.println("Error 2nd redire
                        throw new IOException("Error url " +
                                              " to 2nd redirect ur
                }
                m_redirect = true;
                m_redirectUrl = url;
                com.substanceofcode.rssreader.businessentit
                                HTMLLinkParser.parseFeeds(ne


//#
//#
//#
```

- Spotlight



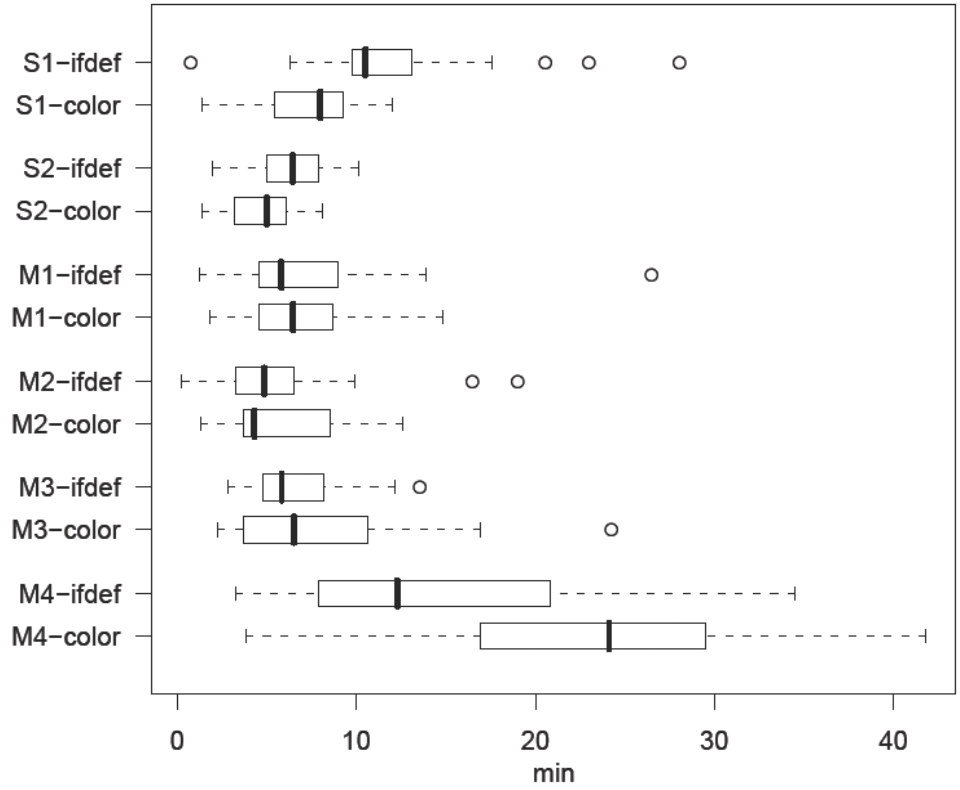
Scaling Visual Representations

- Focus on few features at a time
- Repeating colors / manual assignment sufficient
- Analysis of 4 Java ME and 40 C programs:
  - 96 % pages of source code with ≤ 3 colors
  - 99 % pages of source code with ≤ 7 colors



Program Comprehension: An Experiment

- #ifdef vs. colors
- 43 subjects in 2 groups
- S1-2: search tasks faster with colors (43% & 23%)
- M1-3: maintenance tasks same perform.
- M4: maintenance task with red back-ground color -37%
- No influence on correctness
- Subjects prefer colors



Agenda

- Problems and Advantages of Preprocessors
- 4 Improvements
  - Views
  - Visual Representation
  - Disciplined Annotations
  - Product-Line-Aware Type System
- Summary and Perspective

[TOOLS'09]

Disciplined Annotations

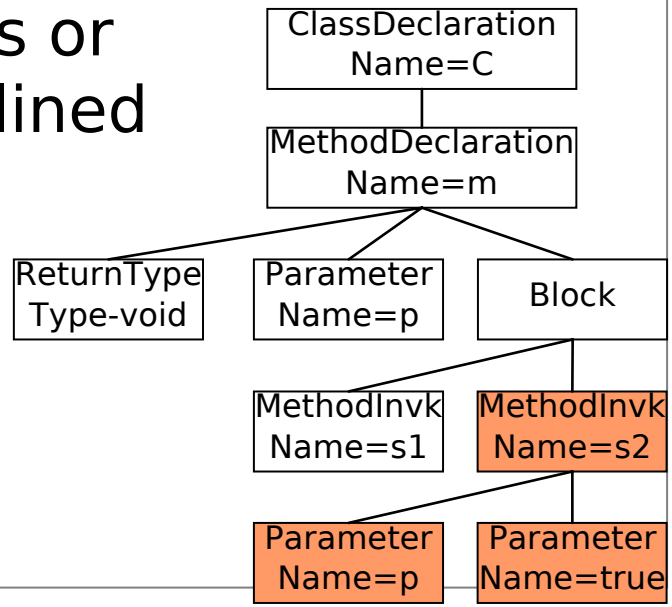
- Syntax errors caused by annotations on plain text

```
class Foo { }
```

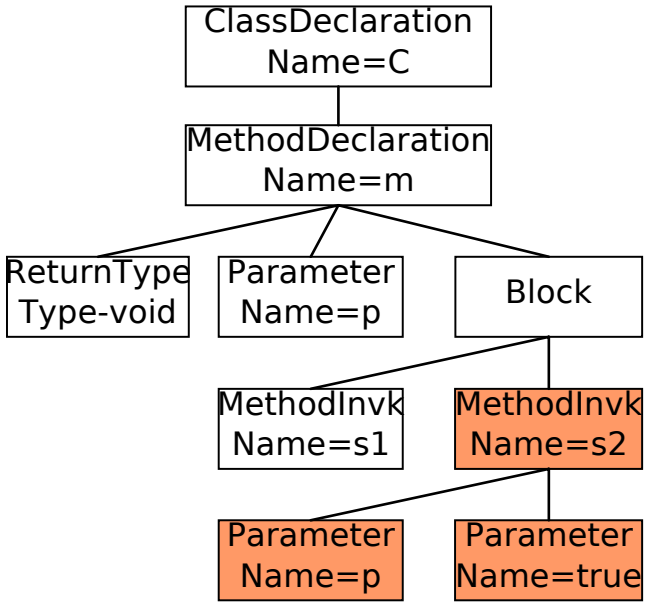
```
class Foo { }
```

- Solution: Use underlying artifact structure
  - Enforce disciplined annotations on entire classes, methods, or statements
  - Annotations of isolated brackets or keywords regarded as undisciplined

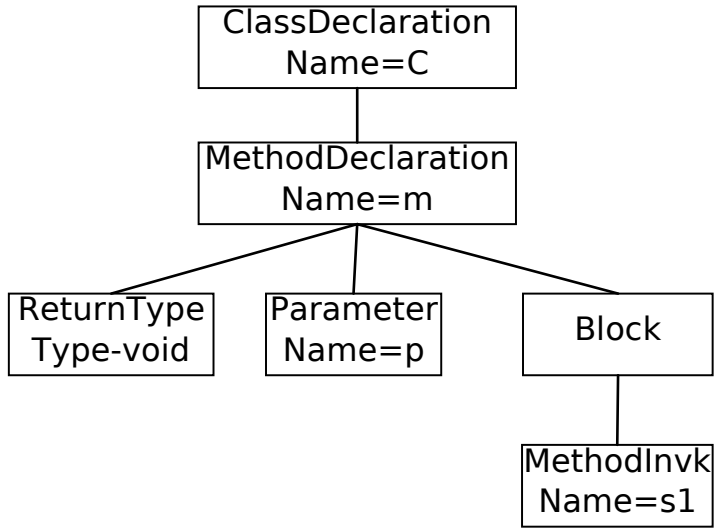
```
class C {
    void m(int p) {
        s1();
        s2(p, true);
    }
}
```



Variant Generation by AST Transformation



variant generation



parsing

print / unparse

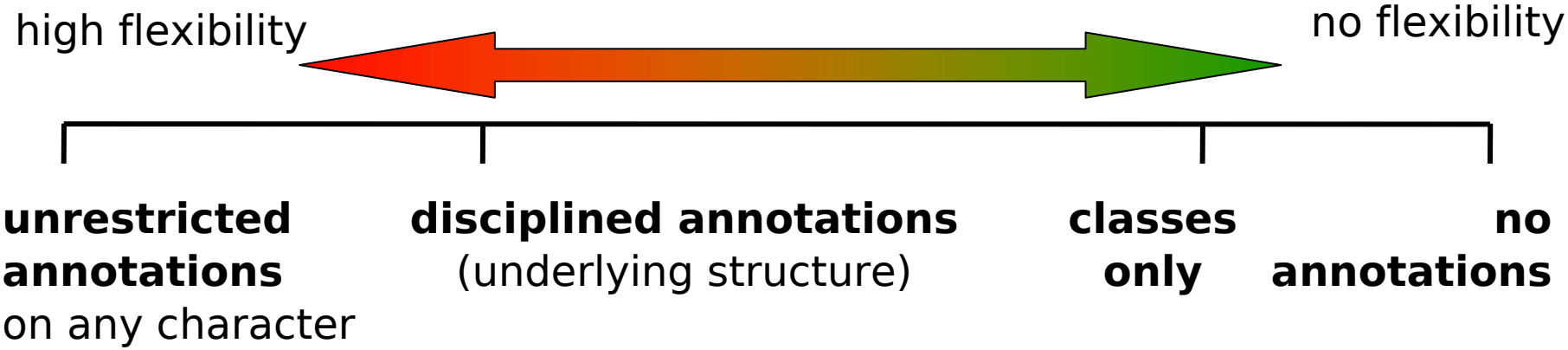
```
class C {
    void m(int p) {
        s1();
        s2(p, true);
    }
}
```

```
class C {
    void m(int p) {
        s1();
    }
}
```

Variant generation cannot cause syntax errors.

Expressiveness of Disciplined Annotations

- *Not all annotations are possible*
- Sometimes workarounds required
- Experience: not a significant restriction in practice
- ~90% of all annotations in 40 C programs is disciplined already



Agenda

- Problems and Advantages of Preprocessors
- 4 Improvements
  - Views
  - Visual Representation
  - Disciplined Annotations
  - Product-Line-Aware Type System
- Summary and Perspective

[ASE'08]

Product-Line-Aware Type System

- Base: Type correct without annotations (for tool support)
- Detecting all pairs:
  - Method invocation and method declaration
  - Reference and declaration of class, variable, etc.
  - ...
- Comparison of annotations for each pair

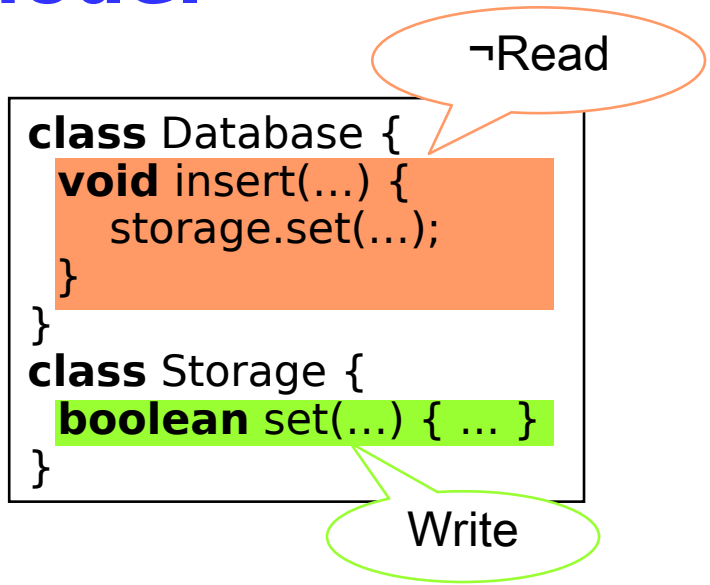
```
class Database {
  Storage storage;
  void insert(Object data
              Txn txn) {
    storage.set(data,
                txn.getLock())
  }
}
class Storage {
#ifdef WRITE
   boolean set(…) { ... }
#endif
}
```

Declaration annotated + Reference not annotated → Error in some variants

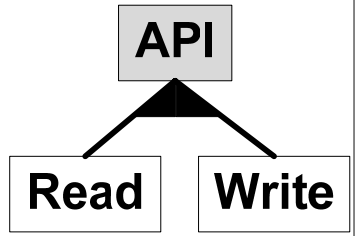
Related Work:
- SafeGen
- fmp2rsm
- Safe Composition
- FFJPL
- Conditional Methods

Type Checking in the Context of a Feature Model

- Handling dependencies between features
- Can all variants with reference reach a corresp. declaration?



$$FM = API \wedge (API \Rightarrow (read \vee write))$$



- Implementation: Propositional logic and SAT solvers:

In all variants in which code fragment A is included also code fragment B is included:

$$FM \Rightarrow (AT(a) \Rightarrow AT(b))$$

[ASE'08]

Formalization: CFJ

- On top of Featherweight Java
- Theorem: Generation preserves typing
  - Formal proof with Coq

Term typing $\boxed{\Gamma \vdash t : C}$

$$\frac{x : C \text{ with } \mathcal{A}' \in \Gamma \quad \mathcal{A} \to \mathcal{A}'}{\mathcal{A};\Gamma \vdash x : C} \text{ (T-VAR)}$$

$$\frac{\mathcal{A};\Gamma \vdash t_0 : C_0 \quad fields(C_0) = \overline{C\ f} \quad \mathcal{A} \to AT(C_i\ f_i)}{\mathcal{A};\Gamma \vdash t_0.f_i : C_i} \text{ (T-FIELD)}$$

$$\frac{\mathcal{A};\Gamma \vdash t_0 : C_0 \quad mtype(m, C_0, \mathcal{A}) = \overline{D\ y} \to C \quad AT(\bar{t});\Gamma \vdash \bar{t} : \overline{C} \quad \overline{C} <: \overline{D} \quad \mathcal{A} \to (AT(\bar{t}) \leftrightarrow AT(\overline{D\ y})) \quad AT(\bar{t}) \to \mathcal{A}}{\mathcal{A};\Gamma \vdash t_0.m(\bar{t}) : C} \text{ (T-INVK)}$$

$$\frac{fields(C) = \overline{D\ f} \quad AT(\bar{t});\Gamma \vdash \bar{t} : \overline{C} \quad \overline{C} <: \overline{D} \quad \mathcal{A} \to AT(C) \quad \mathcal{A} \to (AT(\bar{t}) \leftrightarrow AT(\overline{D\ f})) \quad AT(\bar{t}) \to \mathcal{A}}{\mathcal{A};\Gamma \vdash \text{new } C(\bar{t}) : C} \text{ (T-NEW)}$$

$$\frac{\mathcal{A};\Gamma \vdash t_0 : D \quad D <: C}{\mathcal{A};\Gamma \vdash (C)t_0 : C} \text{ (T-UCAST)}$$

$$\frac{\mathcal{A};\Gamma \vdash t_0 : D \quad C <: D \quad C \neq D \quad \mathcal{A} \to AT(C)}{\mathcal{A};\Gamma \vdash (C)t_0 : C} \text{ (T-DCAST)}$$

$$\frac{\mathcal{A};\Gamma \vdash t_0 : D \quad C \not<: D \quad D \not<: C \quad \textit{stupid w.}}{\mathcal{A};\Gamma \vdash (C)t_0 : C} \text{ (T-SCAST)}$$

$$\frac{P_{CFJ}\ \text{OK} \quad F \text{ is valid}}{variant(P_{CFJ}, F)\ \text{OK}}$$

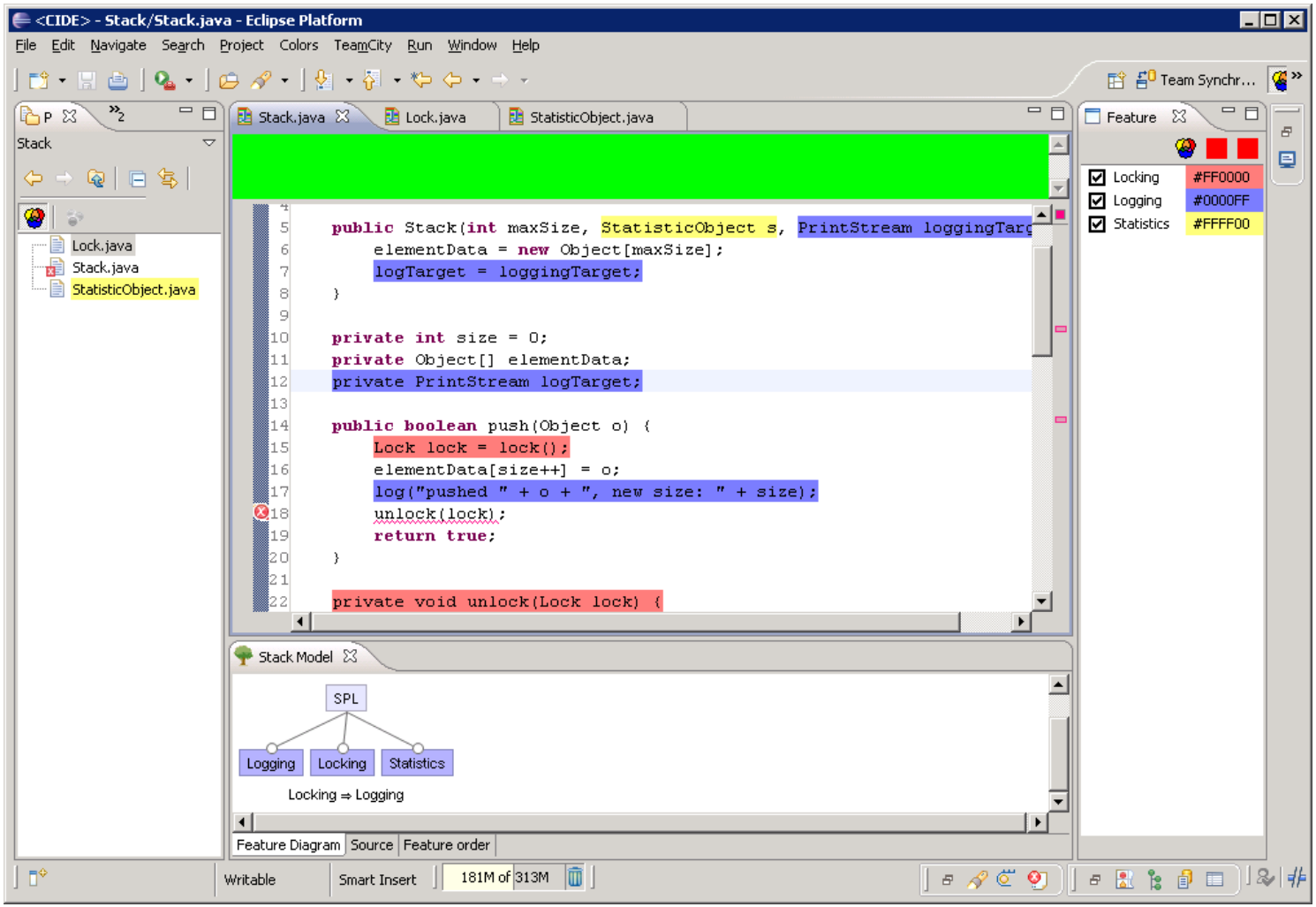
Method typing $\boxed{\text{M OK in C}}$

$$\frac{\begin{array}{c}M = C_0\ m(\overline{C\ x})\ \{\text{return } t_0;\} \quad AT(M) = \mathcal{A} \quad \mathcal{A} \to AT(C_0) \quad AT(\overline{C\ x}) \to AT(\overline{C}) \\ CT(C) = \text{class C extends D } \{\ldots\} \quad override(m, D, \overline{C} \to C_0, \mathcal{A}) \\ \Gamma = \overline{x} : \overline{C} \text{ with } AT(\overline{C\ x}), \text{this} : C \text{ with } AT(C) \quad \mathcal{A};\Gamma \vdash t_0 : E_0 \quad E_0 <: C_0 \quad AT(\overline{C\ x}) \to \mathcal{A}\end{array}}{\text{M OK in C}} \text{ (T-METHOD)}$$

Class typing $\boxed{\text{C OK}}$

$$\frac{\begin{array}{c}K = C(\overline{D\ g}, \overline{C\ f}')\ \{\text{super}(\overline{g}'); \overline{\text{this.f=f;}}\} \quad \overline{M}\ \text{OK in C} \quad fields(D) = \overline{D\ g}'' \\ \overline{C\ f} = \overline{C\ f}' \quad \overline{D\ g} = \overline{D\ g}'' \quad \overline{g} = \overline{g}' \quad AT(C) = \mathcal{A} \quad \mathcal{A} \to AT(D) \\ AT(\overline{C\ f}) \leftrightarrow AT(\overline{\text{this.f=f}}) \quad AT(\overline{C\ f}) \leftrightarrow AT(\overline{C\ f}') \quad \mathcal{A} \to (AT(\overline{D\ g}) \leftrightarrow AT(\overline{D\ g}'')) \\ AT(\overline{D\ g}) \leftrightarrow AT(\overline{g}') \quad AT(\overline{C\ f}) \to AT(\overline{C}) \quad AT(\overline{C\ f}) \to \mathcal{A} \quad AT(\overline{M}) \to \mathcal{A} \quad AT(\overline{D\ g}) \to \mathcal{A}\end{array}}{\text{class C extends D } \{\ \overline{C\ f};\ K\ \overline{M}\ \}\ \text{OK}} \text{ (T-CLASS)}$$

Implementation: CIDE



http://fosd.de/cide

[GPCE'09]

Automated Refactorings

- Feature Module

Base
```
class Stack {
    int[] data;
    void push(int o) { … }
    int pop() { /*...*/ }
}
```

Top
```
refines class Stac
    int top() { /
}
```

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

Automated Transformation

- Annotationen

```
class Stack {
    int[] data;
#ifdef UNDO
    int backup;
    void undo() { /*...*/ }
#endif
    P
    () { /*...*/ }
#endif
    void push(int o) {
#ifdef UNDO
        backup = top();
#endif
        /*...*/
    }
    int pop() { /*...*/ }
}
```

Agenda

- Problems and Advantages of Preprocessors
- 4 Improvements
  - Views
  - Visual Representation
  - Disciplined Annotations
  - Product-Line-Aware Type System
- Summary and Perspective

Summary

- Problems:
  - Separation of Concerns
  - Obfuscation
  - Error-proneness
- Retained advantages:
  - Easy to use
  - Flexible
  - Language-independent
  - Low adoption barrier
- Improvements:
  - Views
  - Visual representation
  - Disciplined annotations
  - Product-line-aware type system

“Virtual Separation of Concerns”

Summary

- Problems:
  - Separation of Concerns
  - Obfuscation
  - Error-proneness
- Improvements:
  - Views
  - Visual representation
  - Disciplined annotations
  - Product-line-aware type system
- Retained advantages:
  - Easy to use
  - Flexible
  - Language-independent
  - Low adoption barrier

“Virtual Separation of Concerns”

Summary

- Problems:
  - Separation of Concerns
  - Obfuscation
  - Error-proneness
- Improvements:
  - Views
  - Visual representation
  - Disciplined annotations
  - Product-line-aware type system
- Retained advantages:
  - Easy to use
  - Flexible
  - Language-independent
  - Low adoption barrier
- Remaining Problems:
  - Separate Compilation
  - Black-box Reuse

“Virtual Separation of Concerns”

Perspective

- Preprocessors common in practice, despite strong criticism
- Neglected by researchers
- Tool support can address most problems

- Interim solution (with migration path) or serious alternative?

- Give preprocessors a second chance!

Questions?

http://fosd.de/cide