3.36pt

Featureous: An Integrated Approach To Location, Analysis And Modularization Of Features In Java Applications (SB5-MAI)

Jan Corfixen Sørensen
and Andrzej Olszak

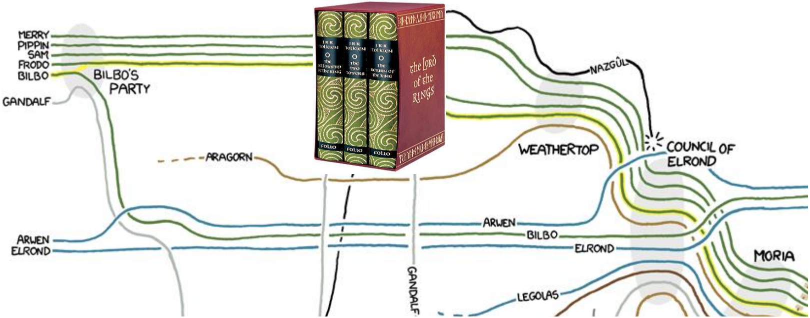
University of Southern Denmark

August 30, 2017

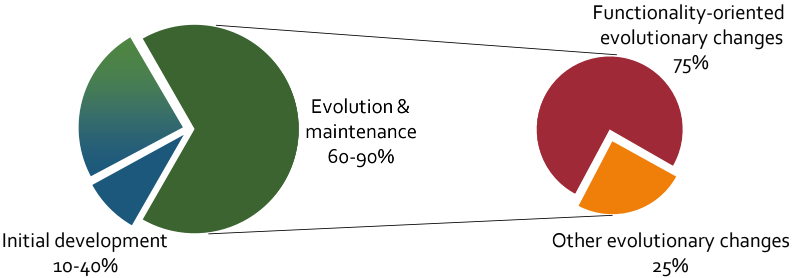
Motivation

The Role of Modularity

- There are various criteria for dividing software into modules
- 'Proper' modularization facilitates:
  - *Comprehension:* understanding systems one module at a time
  - *Change:* modifying modules independently
  - *Work division:* dividing work on modules boundaries

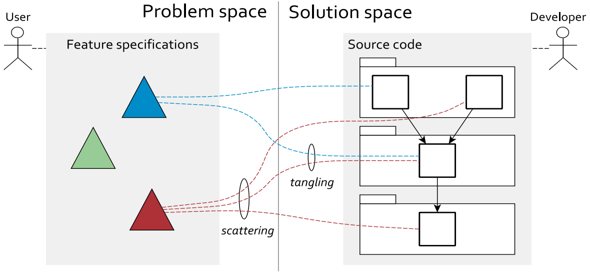


The Role of Features



- *Feature* – unit of user-identifiable functionality of software
- Feature-oriented change in nutshell:
  - User ⇨ Request ⇨ Developer ⇨ Code

Features in OO software

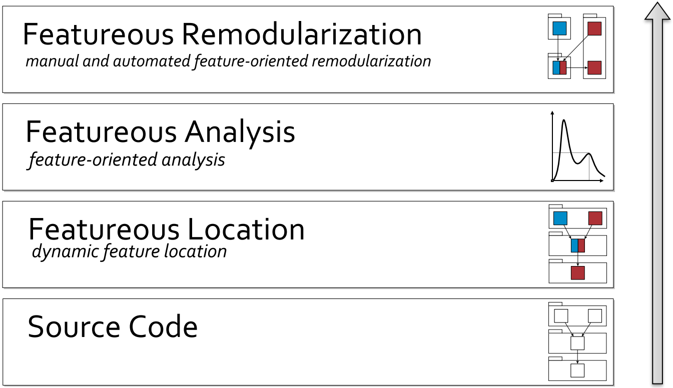


- Features as inter-class collaborations:
  - Implicit mappings and boundaries
  - Scattered (increased change scope and delocalization effects)
  - Tangled (increased change propagation and interleaving effects)

RQ: *How can features of Java applications be located, analyzed and modularized to support comprehension and modification during software evolution?*

Overview of Featureous

Conceptual model of Featureous



Featureous Remodularization

manual and automated feature-oriented remodularization

Featureous Analysis

feature-oriented analysis

Featureous Location

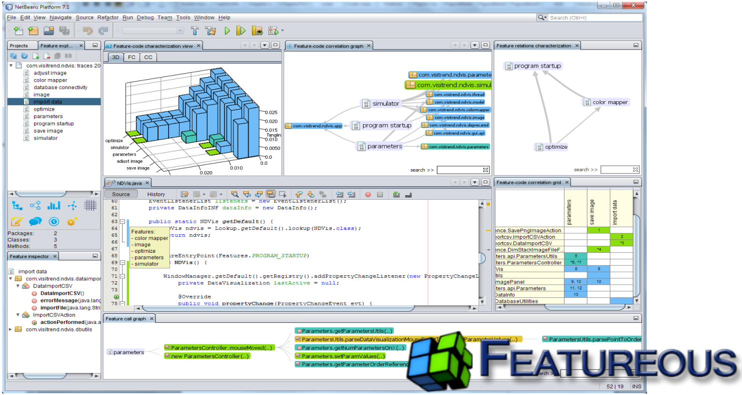
dynamic feature location

Source Code

- Layered conceptual model
  - Incremental design, implementation and evaluation

The Featureous Workbench

- Tool-based approach – implemented as NetBeans plugin
- Applicative studies as evaluation



NetBeans Platform 7.1
File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
Search (Ctrl+I)
Projects
Feature expl...
Feature-code characterization view
3D
FC
CC
optimize
simulator
parameters
adjust image
save image
0.025
0.020
0.015
0.010
0.0050
0.0
0.020
0.010
Feature-code correlation graph
com.visitrend.ndvis.paramete
com.visitrend.ndvis.simul
com.visitrend.ndvis.thread
com.visitrend.ndvis.model
com.visitrend.ndvis.colormapper
com.visitrend.ndvis.image
com.visitrend.ndvis.deprecated
com.visitrend.ndvis.gui.api
com.visitrend.ndvis.parameters
com.visitrend.ndvis.app
simulator
program startup
parameters
search >>
Feature relations characterization
program startup
color mapper
optimize
search >>
NDVis.java
Source
History
Feature-code correlation grid
Feature inspector
Feature call graph
parameters
ParametersController.mouseMoved(..)
new ParametersController(..)
Parameters.getParametersUtils(..)
ParametersUtils.parseDataVisualizationMouse
Parameters.getNumParametersOn(..)
Parameters.setParamValues(..)
Parameters.getParameterOrderReferenc
ParameterValues(..)
ParametersUtils.parsePointToOrder
FEATUREOUS
52 | 19
INS

- com.visitrend.ndvis: traces 20
  - adjust image
  - color mapper
  - database connectivity
  - image
  - import data
  - optimize
  - parameters
  - program startup
  - save image
  - simulator

Packages: 2
Classes: 3
Methods: 5

- import data
  - com.visitrend.ndvis.dataimpor
    - DataImportCSV
      - **DataImportCSV()**
      - **errorMessage**(java.lang
      - **importFile**(java.lang.Stri
    - ImportCSVAction
      - **actionPerformed**(java.a
  - com.visitrend.ndvis.dbutils

```
        EventListenerList listeners = new EventListenerList();
        private DataInfoINF dataInfo = new DataInfo();

        public static NDVis getDefault() {
            Vis ndvis = Lookup.getDefault().lookup(NDVis.class);
            turn ndvis;

            reEntryPoint(Features.PROGRAM_STARTUP)
            NDVis() {

            WindowManager.getDefault().getRegistry().addPropertyChangeListener(new PropertyChangeL
                private DataVisualization lastActive = null;

                @Override
                public void propertyChange(PropertyChangeEvent evt) {
```

Features:
- color mapper
- image
- optimize
- parameters
- simulator

	parameters	save image	import data
ence.SavePngImageAction		1	
portcsv.ImportCSVAction			2
portcsv.DataImportCSV			*3
ence.DimStackImageFileF...		*4	
ters.api.ParametersUtils	5		
ters.ParametersController	*6, *7		
Vis	8	8	
tils			
magePanel	9, 10	10	
ters.api.Parameters	11, 12		
DataInfo	13		
DatabaseUtilities			

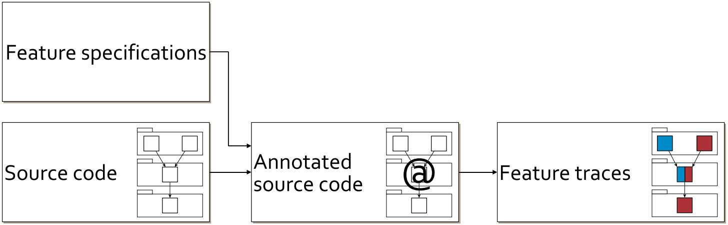
Featureous Location

The Challenge of Feature Location

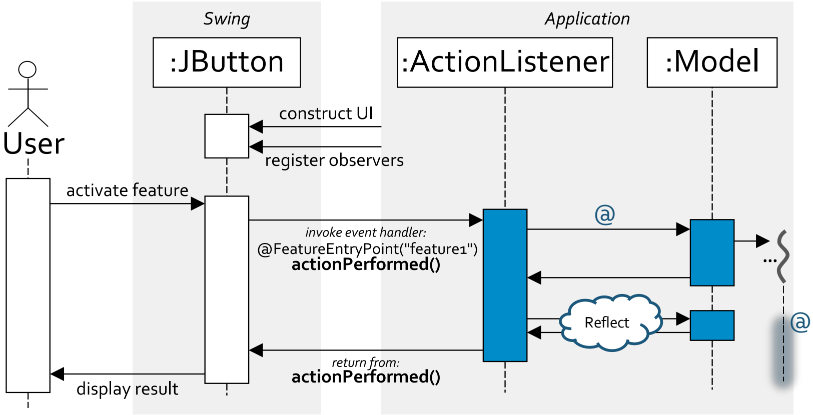
- *Feature location* – identifying source code units that implement features
- Manual approaches
  - Problems with scaling and reproducibility
- Existing semi-automated approaches
  - Require dedicated test suites
  - Rely on artifacts other than source code

The Method

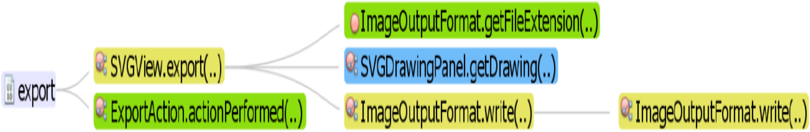
- Dynamic analysis – execution tracing
  - Resolutions of polymorphism and conditionals
- The notion of *feature-entry points*
  - Annotated “entrances to features” to guide tracing



Feature Tracer with AspectJ LTW



Call-Tree


export
SVGView.export(..)
ExportAction.actionPerformed(..)
ImageOutputFormat.getFileExtension(..)
SVGDrawingPanel.getDrawing(..)
ImageOutputFormat.write(..)
ImageOutputFormat.write(..)

Evaluation

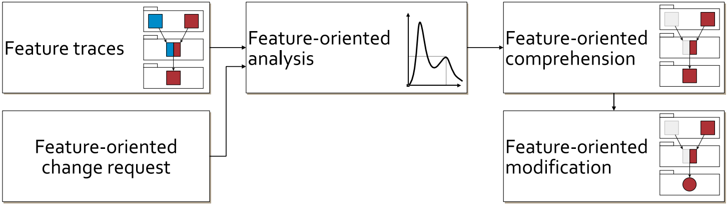
- Applied to 6 medium-sized unfamiliar OSS
  - Discussed cases of BlueJ and JHotDraw SVG

	BlueJ	JHotDraw SVG
Application size	78 KLOC	62 KLOC
Number of identified use cases	127	90
Number of identified features	41	31
Number of feature-entry points	228	91
Class code coverage	**66%**	**75%**
Total time	**8 hours**	**5 hours**

*Estimated manual location time**	*111-195 hours*	*89-155 hours*

Featureous Analysis

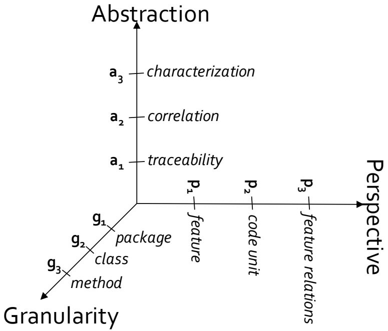
Features as Units of Code Analysis


Feature traces
Feature-oriented change request
Feature-oriented analysis
Feature-oriented comprehension
Feature-oriented modification

- *Feature-oriented analysis* treats features as first-class code investigation entities
- Several metrics and visualizations exist
  - Not always compatible with one another
  - Need firm evolutionary grounding
  - Lack of usable implementations

Structuring Feature-Oriented Analysis

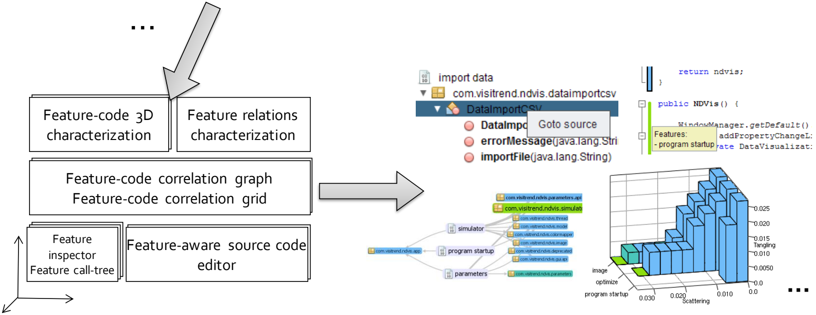
- Unifying conceptual framework for describing views
  - Granularity, Perspective, Abstraction



- Objective definition of views
- 3x3x3 possible configurations

Instantiating the Framework

		Planning phase		Change implementation		
	Request for change	Software comprehension	Change impact analysis	Restructuring for change	Change propagation	Verification and validation
Granularity	n/a	g_2 g_1	g_2 g_1	g_3 g_2 g_1	g_3 g_2	g_3 g_2


...
Feature-code 3D characterization
Feature relations characterization
Feature-code correlation graph
Feature-code correlation grid
Feature inspector
Feature call-tree
Feature-aware source code editor
import data
com.visitrend.ndvis.dataimportcsv
DataImportCSV
DataImpo
Goto source
errorMessage(java.lang.Stri
importFile(java.lang.String)
Features:
- program startup
com.visitrend.ndvis.parameters.api
com.visitrend.ndvis.simulati
simulator
program startup
parameters
com.visitrend.ndvis.app
com.visitrend.ndvis.parameters
image
optimize
program startup
Scattering
Tangling
0.030
0.020
0.010
0.0
0.025
0.020
0.015
0.010
0.0050
0.0
...

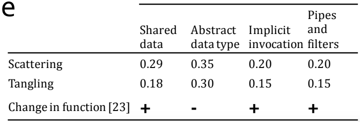
```
return ndvis;
}

public NDVis() {

    WindowManager.getDefault()
        addPropertyChangeL
    ate DataVisualizat
```

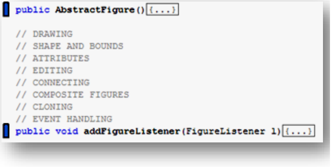
Evaluation

1. Parnas' KWIC textbook example
   - 4 modularization alternatives
   - Results consistent with analyses of Parnas and Garlan



	Shared data	Abstract data type	Implicit invocation	Pipes and filters
Scattering	0.29	0.35	0.20	0.20
Tangling	0.18	0.30	0.15	0.15
Change in function [23]	+	-	+	+

2. Feature-Oriented comprehension of JHotDraw SVG



```
public AbstractFigure() {...}

// DRAWING
// SHAPE AND BOUNDS
// ATTRIBUTES
// EDITING
// CONNECTING
// COMPOSITE FIGURES
// CLONING
// EVENT HANDLING
public void addFigureListener(FigureListener l) {...}
```

3. Adoption of feature-oriented change in JHotDraw SVG
4. Analytical evaluation of support for comprehension with the framework of Storey et al.