



From Feature to Code

SCRUM + NetBeans RCP + Featureous

John Kostaras

JCreate 25-29 August 2014

Agenda

- * SCRUM
- * NetBeans RCP
- * Featureous

SCRUM

What is SCRUM

- * a methodology
- * an agile framework for software development
- * relies on **self-organizing, cross-functional teams**

SCRUM



Steering Group

- Business Conditions & Requirements
- Technology

- Standards
- Conventions
- Guidelines

- Formal Evaluation Criteria



Product Owner

Product Backlog



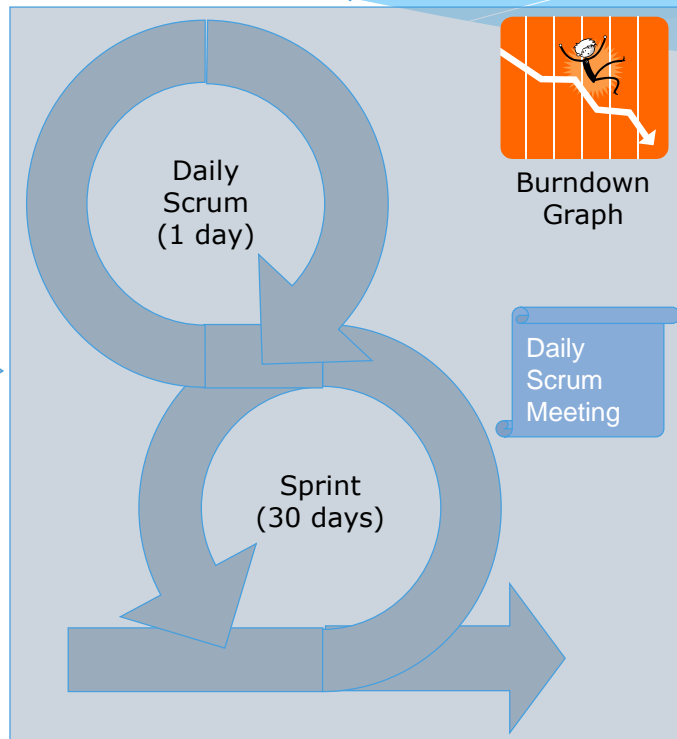
Sprint Planning Meeting



Sprint Backlog



Team



Burndown Graph

Daily Scrum Meeting

• Executable Product Increment



Sprint Review



Impediments List



Scrum Master

Roles

* **SCRUM Team**

- * consists of 5-9 members
- * is self-organized and the members have a joint responsibility for the results

* **SCRUM Master**

- * coaches the development team
- * removes any possible impediments
- * ensures that the team has the best possible circumstances for realizing the goals fixed for the Sprint

Roles (cont.)

* **Product Owner**

- * administers the **Product Backlog** (a.k.a. a todo list) which contains all the specifications, changes and/or functionalities planned for the product and prioritizes them
- * before each Sprint, the highest prioritized goals are transferred to a **Sprint Backlog**

Infrastructure

- * Team (including Scrum Master and Product Owner) should be colocated in one room
- * The development environment must be directly available in the room
- * Test facilities must be near by
- * The room must have a SCRUM board (the bigger the better ;-)
and a small meeting place in front of the board
- * Should include a common build server that can run any automated tests

SCRUM Board

• Product Backlog

• Product Burndown

• Next

• In progress

• Done

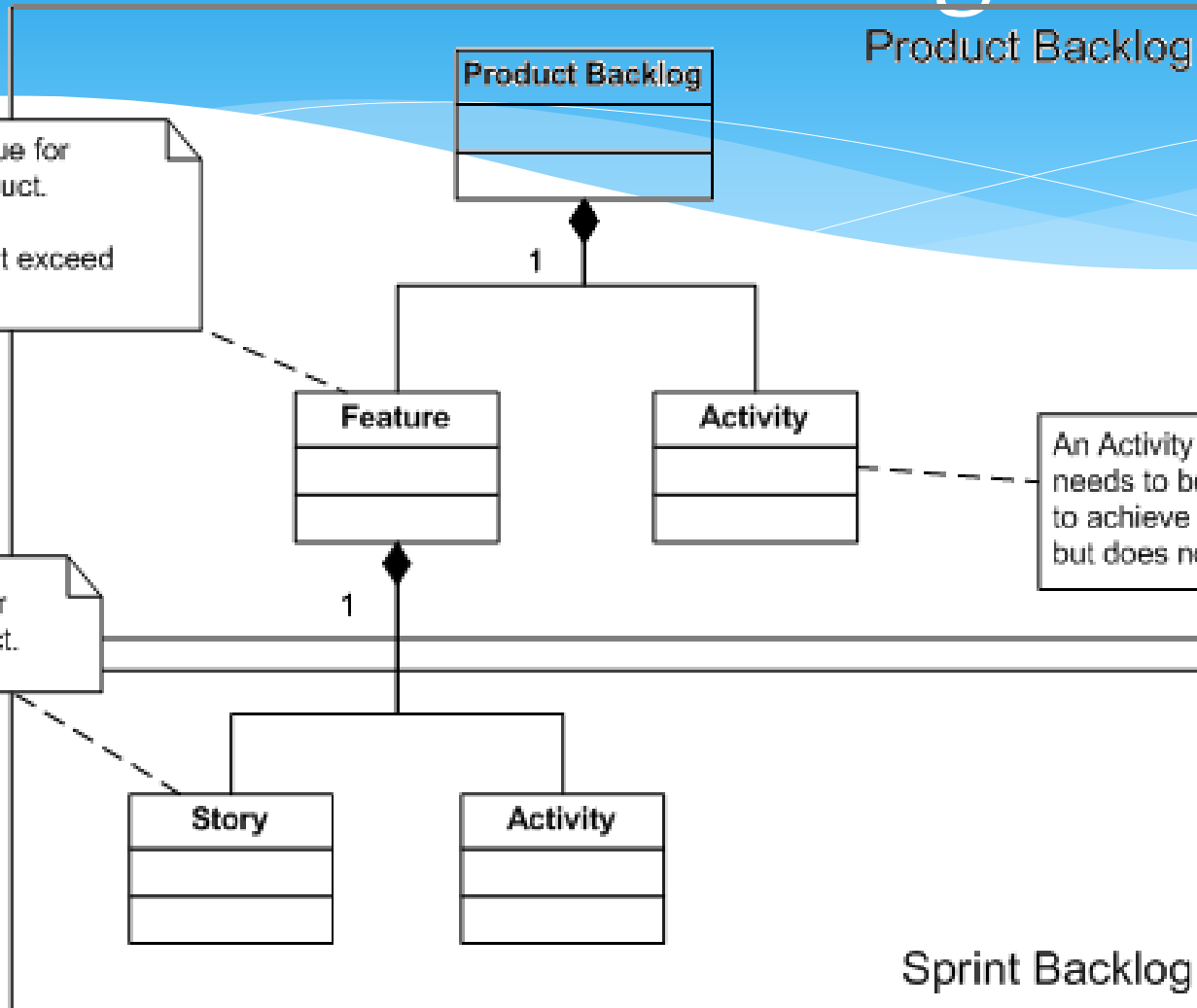
• Impediments

• Sprint Burndown



Burndown
Graph

Product Backlog



Process

- * A FEATURE READY criteria must be defined to decide when a feature is ready to enter a sprint
- * A FEATURE DONE criteria must be defined to decide when a feature is finished – This includes required documentation
- * A STORY DONE criteria must be defined to decide when a story is finished – This includes required documentation
- * Documentation model
 - * Keep it simple;
 - * Design documentation: focus on design decisions and alternatives

Scrum vs UML

SCRUM

- * Feature
- * Story
- * Activity

UML

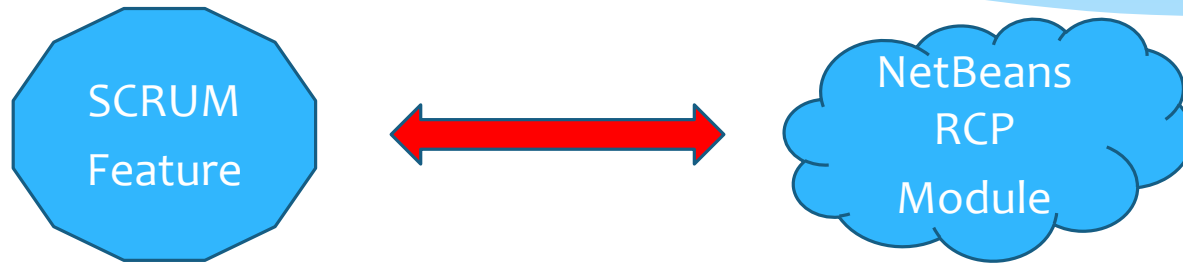
- * Collection of Use Cases
- * Use Case
- * Use Case

NetBeans RCP

What is NetBeans RCP

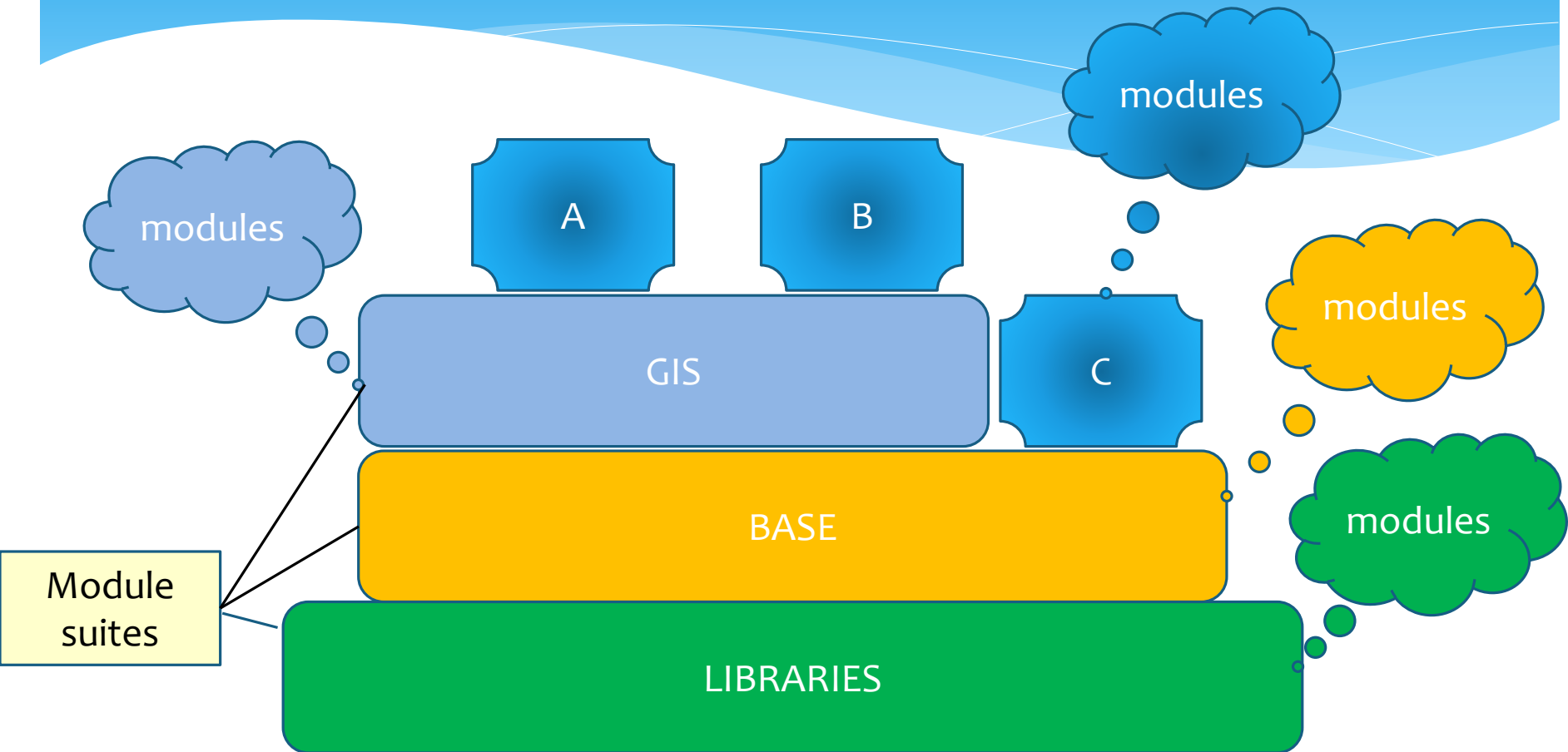
- * Rich Client Platform that NetBeans is build upon
- * Modularised (based on OSGi)

How does it fit to SCRUM?



Not always possible
but wished

A layered architecture



Featureous

What is Featureous

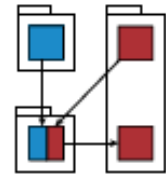
- * A NetBeans plugin
- * Addresses the following question:

***I must modify/fix a bug in a certain feature,
but what exactly are the classes that I should change?***

Conceptual Model

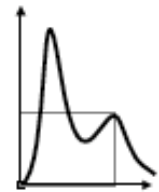
Featureous Remodularization

manual and automated feature-oriented remodularization



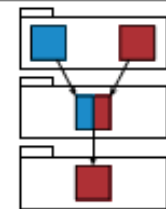
Featureous Analysis

feature-oriented analysis

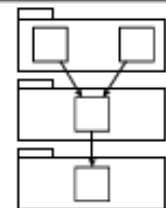


Featureous Location

dynamic feature location



Source Code



Conceptual Model (cont.)

- * **Feature location** establishes traceability links between features and source code of an existing Java application. It is the process of identifying units of source code that contribute to implementing features of an application. This is done via *FeatureEntryPoints*.
- * **Feature-oriented analysis** provides means of visualizing and measuring feature traceability links through several analytical views that provide a theoretically grounded support for comprehension and modification of features.
- * **Feature-oriented remodularization** optimizes the modularization of features in Java package structures. The used method is based on multi-objective optimization of Java package structures. Automated detection of so-called **feature seed methods** is proposed, to enable large-scale feature-oriented quality assessment.

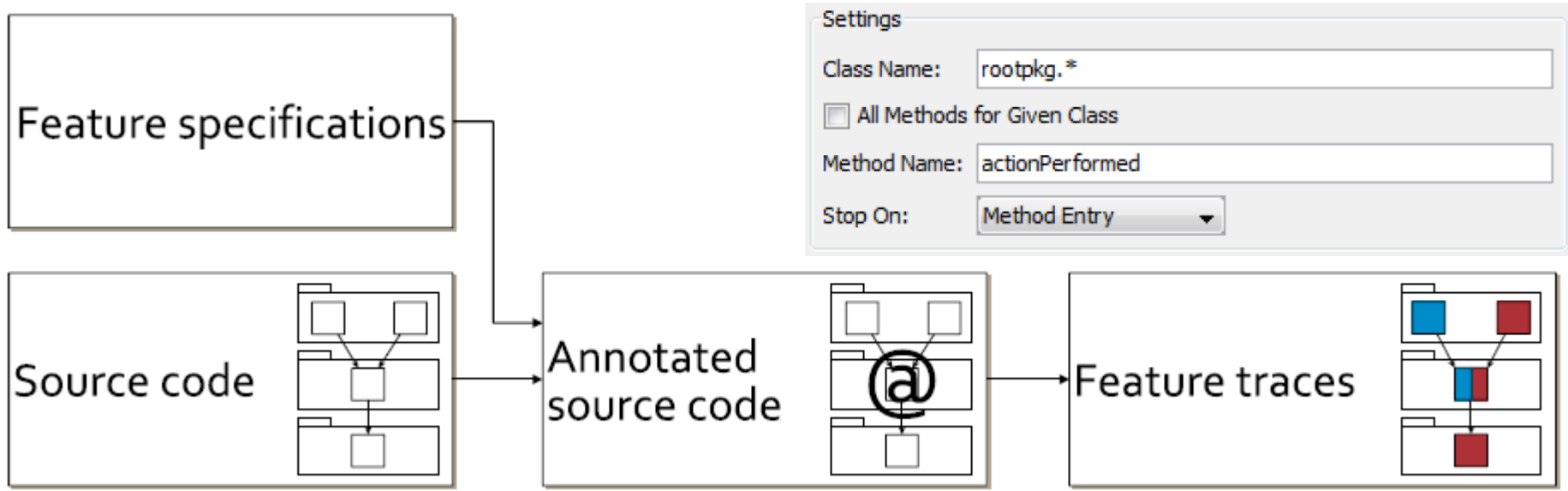
Modes of use

- * Featureous extends the NetBeans IDE with a mechanism for tracing the execution of features
 - * **Feature-centric analysis** investigates correspondences between user-identifiable features and legacy source code
- * The obtained traces can then be measured and investigated from various angles in a number of analytical views (**analysis tool**)
- * Featureous is also a **restructuring tool**; “where is feature X implemented?”
- * The *remodularization workbench* has the ability to automatically relocate classes among packages, according to a set of design objectives (such as coupling and cohesion).
 - * "multi-objective grouping genetic algorithm"

Feature Entry Points

- * For each of the traced *features*, a separate *trace model* is instantiated
- * A feature-trace model captures three types of information about its feature:
 - * the methods and constructors (jointly referred to as Executions) executed,
 - * their enclosing types (i.e. classes, interfaces or enums) and
 - * inter-method invocations
- * a single feature is represented by exactly one, consistent feature-trace model
- * The collected feature-trace models are automatically serialized to files upon the termination of a traced application

Feature Entry Points



* Candidate Feature Entry Points

- * `java.lang.Runnable.run`
- * `java.util.concurrent.Callable.call`
- * `*.main`
- * `java.awt.event.ActionListener.actionPerformed`

Featureous Analysis

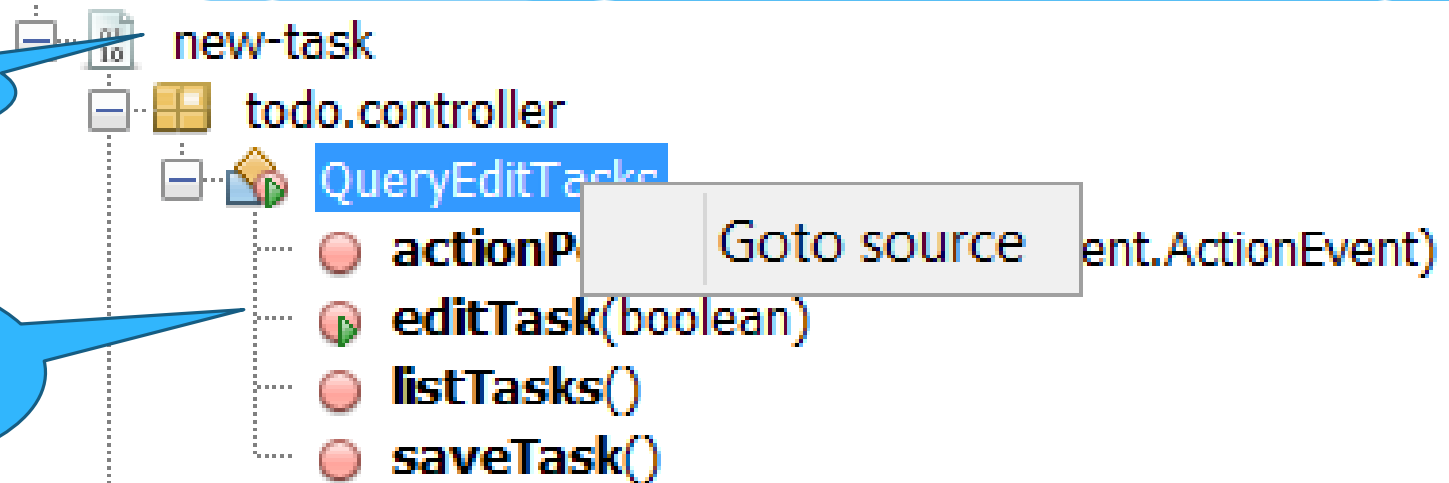
The screenshot displays the NetBeans Platform 7.1 interface with several feature analysis tools open:

- Feature-inspector (1):** Shows a tree view of features for the project 'com.visitrend.ndvis'. The 'import data' feature is selected.
- Feature-codes characterization view (6):** A 3D bar chart showing the distribution of code across features. The x-axis represents features like 'optimize', 'simulator', 'parameter', 'adjust image', and 'save image'. The y-axis represents a numerical value from 0.0 to 0.025.
- Feature-codes correlation graph (4):** A dependency graph showing relationships between features and their underlying code classes. 'com.visitrend.ndvis.app' is the root, with dependencies on 'simulator', 'program startup', and 'parameters'. 'simulator' depends on several classes including 'com.visitrend.ndvis.thread', 'com.visitrend.ndvis.model', 'com.visitrend.ndvis.colormapper', 'com.visitrend.ndvis.image', 'com.visitrend.ndvis.deprecated', and 'com.visitrend.ndvis.guiapi'. 'program startup' depends on 'com.visitrend.ndvis.parameters'.
- Feature relations characterization (7):** A graph showing relationships between features. 'program startup' is connected to 'color mapper' and 'optimize'. 'color mapper' is also connected to 'optimize'.
- Source (3):** Shows the source code for 'NDVis.java'. The code includes an event listener list, data info, and a public static method 'getDefault()' that returns an instance of 'NDVis'. The 'Features' list in the editor includes 'color mapper', 'image', 'optimize', 'parameters', and 'simulator'.
- Feature-code correlation grid (5):** A table showing the correlation between features and code classes. The columns are 'parameters', 'save image', and 'import data'. The rows are 'SavePngImageAction', 'ImportCSVAction', 'DataImportCSV', 'DimStackImageFile', 'ParametersUtils', 'ParametersController', 'Vis', 'Vis', 'ImagePanel', 'Parameters', 'DataInfo', and 'DatabaseUtilities'. The grid shows numerical values in the cells, such as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, and 13.
- Feature call graph (2):** A graph showing the call relationships between features. 'parameters' is the root, with calls to 'ParametersController.mouseMoved(...)', 'ParametersUtils.parseDataVisualizationMouseEventToOrderedParameterValues(...)', 'ParametersUtils.parseIntToOrder', 'Parameters.getNumParametersOnX(...)', 'Parameters.setParamValues(...)', and 'Parameters.getParameterOrderReference(...)'. 'ParametersController.mouseMoved(...)' also calls 'ParametersUtils.parseIntToOrder'.

Feature Inspector

feature

Feature
Entry point



- * provides detailed navigable traceability links from features to concrete fragments of an application's source code

Feature Call Graph

feature

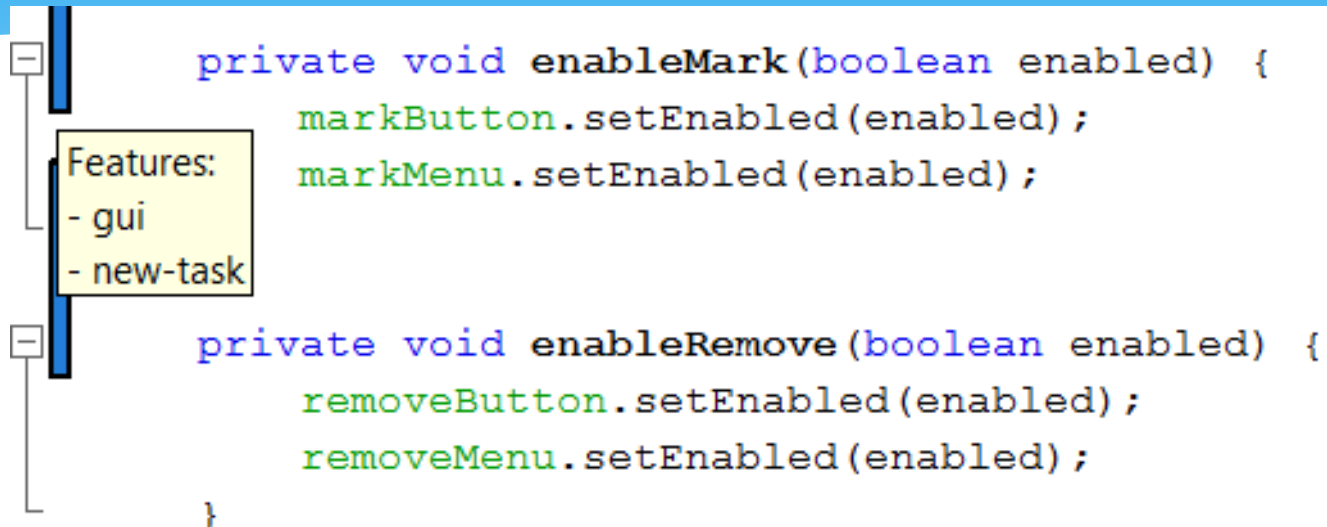
Green means single-feature
source code units

 new-task

 QueryEditTasks.editTask(..)

- * provides a tree-based visualization of the runtime call graphs of methods implementing features
- * edges among method-nodes stand for inter-method *call*-relations registered at runtime

Feature-Aware Source Code Editor



```
private void enableMark(boolean enabled) {
    markButton.setEnabled(enabled);
    markMenu.setEnabled(enabled);
}

private void enableRemove(boolean enabled) {
    removeButton.setEnabled(enabled);
    removeMenu.setEnabled(enabled);
}
```

- * extends the default source code editor of the NetBeans IDE with automated feature-oriented code folding and a colored sidebar visualizing traceability from code to features
- * Tooltips display the feature names

Feature-Code Correlation Graph



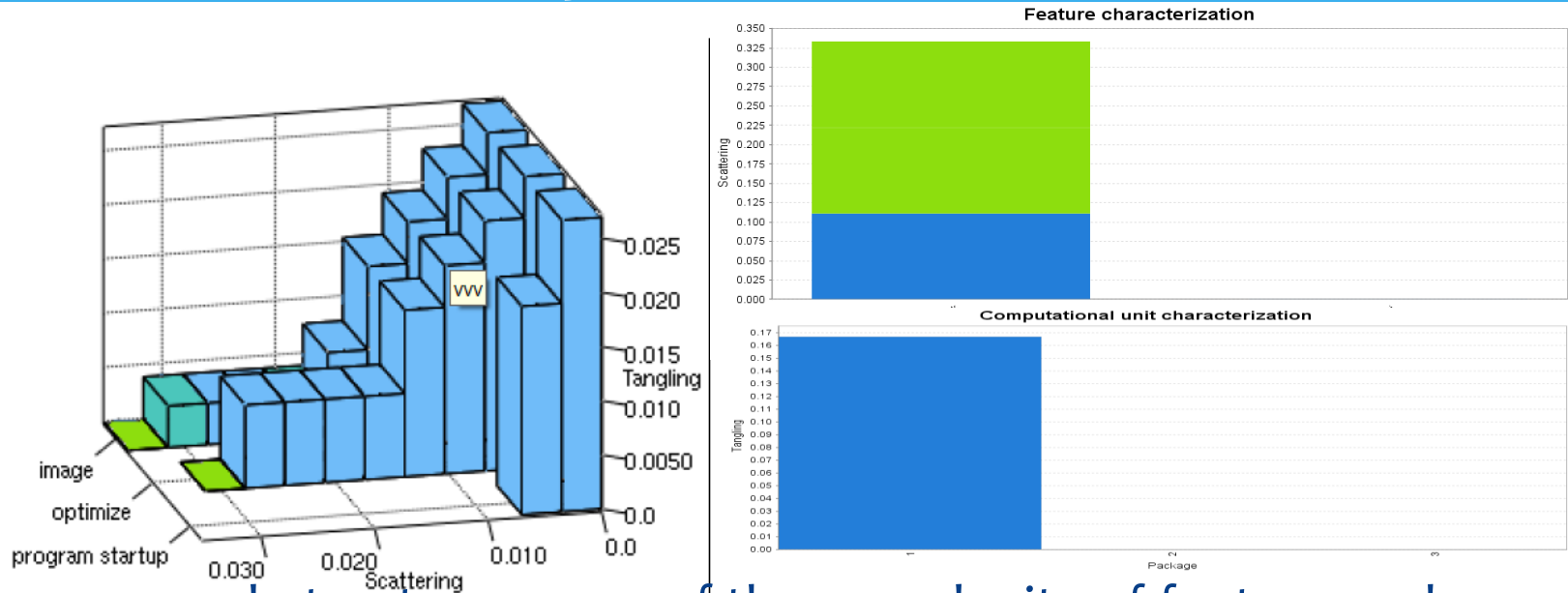
- * a graph-based visualization of traceability links (edges) between features (nodes) and classes or packages (nodes)
- * edge-thickness-based indicates the number of contained source code units that participate in the traceability relations with individual features
- * this view can be used to depict scattering and tangling of features

Feature-Code Correlation Grid

- * correlates features with packages or classes using a matrix-based representation.
- * displays the concrete identifiers of the objects of a given class used by features. The sorting of the matrix's rows and columns is done according to the scattering of features and tangling of code units.

| | new-task | gui |
|-----------------|----------|-----|
| todo.model | | |
| todo.controller | | |
| todo.view | | |

Feature-Code 3D Characterisation



- * serves as an abstract summary of the complexity of feature-code relations.
- * each feature is depicted as a series of bars
- * each bar represents a single package or class used by a feature
- * number of bars displayed for each feature reflects the scattering
- * height of individual bars represents tangling of their corresponding code units

Feature Relations Characterization



- * relates features to each other with respect to how they depend on the objects created by other features and how they exchange data by sharing these objects with one another
- * a directed, solid edge is drawn if a pair of features is in a producer-consumer relation, meaning that objects that are instantiated in one of them are used in another

How to add Featureous

* [Featureous.org](http://featureous.org) says:

1. In your IDE, go to Tools / Plugins / Settings tab
2. Press the Add button to add an update site to configuration
3. Insert **<http://featureous.org/update-sites/src-utils/updates.xml>** in the URL field
4. Add another update site for **<http://featureous.org/update-sites/featureous/updates.xml>**
5. Go to the Available Plugins Check tab and install all the plugins whose names start with "Featureous".
6. Follow the instructions and read the license agreement
7. Restart your IDE, and start Featureous by executing Window → Other → Show Featureous menu action.

How to add Featureous plugin

- * Download the source code:

```
# Non-members may check out a read-only working copy  
anonymously over HTTP.
```

svn checkout

```
http://featureous.googlecode.com/svn/trunk/ featureous-  
read-only
```

- * Open the *Featureous tool* and *Source utils* module suites in NetBeans
- * Fix any compilation errors
- * Right-click on each module suite and choose Package as → NBMs
- * Tools | Plugins | Downloaded tab | Add plugins | Install
- * Tools | Plugins | Installed | Activate User Installed Plugins

Fix compilation errors

* Some TopComponent IDs

```
dk.sdu.mmmi.featureous.remodularization.transform.  
MoveClassNB
```

```
this.refactoring.getContext().add(  
RetoucheUtils.getClasspathInfoFor(  
javaObject.getPrimaryFile()));
```

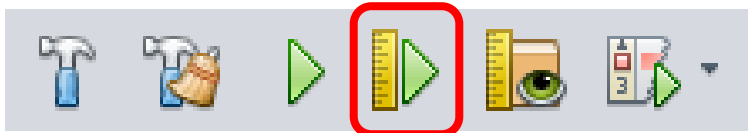


```
this.refactoring.getContext().add(  
ClasspathInfo.create(javaObject.getPrimaryFile()));
```



How to use

- * Select the project in the Explorer View
- * Press the **Trace project** button



- * `dk.sdu.mmmi.featuretracer.xml` is created
- * Edit this file to configure the tracer before you can trace features of your project
 - * replace all
`"/dk\.sdu\.mmmi\.featuretracer\.test\. .* /"` with
your root package e.g.
`"/org\.mycompany\.myproject\. .* /"`

How to use

* Feature location

- * For each feature of a program, a programmer has to annotate its entry points – the methods through which a program's control flow enters implementation of a feature
- * Annotate the methods, e.g. `@FeatureEntryPoint("gui")`
`@FeatureEntryPoint("alerts")` etc.
- * **Trace project** again and try to execute all stories of the feature on the running app as a normal use would do
- * traces are written to `FeatureTraces\traces timestamp`
 - * After annotating feature-entry points, Featureous instruments a program with a tracing aspect at load-time that identifies methods, classes and objects used by individual features at run-time. Feature-trace files serve as an input to further analysis.

How to use

- * Window → Other → Show Featureous
 - * Feature Explorer
 - * Feature Inspector
- * **Update traces** button

How to use in an RCP project

1. Add `ft.jar` and `aspectjweaver.jar` to either your Libraries module of your module suite (right-click → Properties → Libraries → Wrapped JARs → Add JAR) or to the libraries of your module.
2. Annotate feature entry points with `@FeatureEntryPoint`
3. Place an `aop.xml` config file in `src/META-INF` directory of each of the modules that you want to trace. For each of the modules, define the root package that encloses all the classes and packages that you want in the tracing scope.

How to use in an RCP project (cont.)

4. Add `ft.jar` and `aspectjweaver.jar` to the root directory of your module suite.
5. Add the following line to `project.properties` of your module suite:

```
run.args.extra = -J-javaagent:./aspectjweaver.jar  
-cp:a ./ft.jar
```

6. Clean and build the project suite and execute from within the IDE

Work to be done

Doesn't work for RCP

Recap

- * **Scrum** is an agile methodology used to more and more software projects
 - * Consists of Features, Stories and Activities
- * **NetBeans RCP** is a platform that supports breaking up your monolithic application into modules
 - * Suggestion: Implement each Scrum feature to its own module if possible
- * **Featureous** is a NB plugin that allows you to find features of a monolithic application
 - * can be used to trace features of your application
 - * doesn't support RCP applications

References

- * <http://featureous.org>
- * Olszak A. (2012), *Featureous: An Integrated Approach to Location, Analysis and Modularization of Features in Java Applications*, Ph.D. Thesis in Software Engineering, University of Southern Denmark.
- * Olszak A. (2011), “[Modularizing Features of Your Legacy Application for Fun & Profit](#)”
- * Olszak A. (2011), “[Featureous 3.0: Automated Restructuring of Code](#)”
- * Olszak A. & Jorgensen B. N. (2011), “Understanding Legacy Features with Featureous”, IEEE Computer Society, 1095-1350/11.

Questions

