Static Code Analysis

Jakub Papcun
Jan Svoboda

Honeywell

Content

- Honeywell
- Static Code Analysis
- Use of SCA in Honeywell
- Defect Tracking Integration
- Manual Code Review Integration

Honeywell

- **4 Strategic Business Groups (SBGs)**
  - Aero
  - Automation and Control Solutions (ACS)
  - Performance Materials and Technologies (PMT)
  - Transportation Systems (TS)

- **130 000 employees worldwide**

- **ACS Centre of Technologies (ACT)**
  - Software Excellence Group
  - Implements Best Practices for software development
  - Implements tools necessary for fulfilling Best Practices requirements

Static Code Analysis

Honeywell

- **Analysis of the code without executing the program itself**

- **Various types of SCA**
  - **Type checking**
    - checks for correct assignment of types of objects
  - **Style checking**
    - checks the style of the code and its formatting
  - **Program Understanding**
    - helps user make sense of large codebase and may include refactoring capabilities
  - **Program verification and property checking**
    - attempts to prove that the code correctly implements the specification of the program
  - **Security review**
    - uses dataflow analysis for detection of possible code injection
  - **Bug finding**
    - looks for places in the code where program may behave in a different way from the way intended by developer

Static Code Analysis - Limitations

Honeywell

- **SCA identifies only “shallow” errors and does not look for problems in design or functionality**

- **3 types of results**
  - **True positives**
    - real issues which are code errors and should be fixed before releasing
  - **False positives**
    - issues identified by the analysis but not real threats due to for example architecture of the software
  - **False negatives**
    - real issues which Static Code Analysis did not identify and are still hidden from the knowledge of the developers

- **Possibility to adjust the Static Code Analysis rules to the context**

Example 1

Honeywell

```
private Map<String, String> paths = new HashMap<String, String>();

public void addPath(String name, String path) {
    paths.put(name, path);
}

private String getNormalizedPath(String name) throws IOException {
    return paths.get(name).toLowerCase();
}
```

Example 1

Honeywell

```
private Map<String, String> paths = new HashMap<String, String>();

public void addPath(String name, String path) {
    paths.put(name, path);
}

private String getNormalizedPath(String name) throws IOException {
    return paths.get(name).toLowerCase();
}
```

Can return `null`

A NullPointerException is thrown in case of an attempt to dereference a `null` value.

Example 2

Honeywell

```
private static void foo(){
    int i = 0;
    String s = null;

    if(i > 0){
        s = "positive";
    }

    if(s.contains("pos")){
        System.out.println(s);
    }
}
```

Example 2

Honeywell

```
private static void foo(){
    int i = 0;
    String s = null;

    if(i > 0){
        s = "positive";
    }

    if(s.contains("pos")){
        System.out.println(s);
    }
}
```

Statement always `true`

1. Statement is always false and never enters the block

Example 2

Honeywell

```
private static void foo(){
    int i = 0;
    String s = null;

    if(i > 0){
        s = "positive";
    }

    if(s.contains("pos")){
        System.out.println(s);
    }
}
```

Statement always `true`

`s` may be `null`

1. Statement is always false and never enters the block
2. s variable may be null and NullPointerException may be thrown

Example 3

Honeywell

```
private static void foo(int arr[]){
    if(arr != null & arr.length != 0){
      foo2();
    }
    return;
}
```

Example 3

Honeywell

```
private static void foo(int arr[]){
    if(arr != null & arr.length != 0){
      foo2();
    }
    return;
}
```

& or &&

Questionable use of bit operation '&' in expression. Did you mean '&&'?

Example 4

Honeywell

```
private static void foo(int j){
    Integer k;
    switch(k){
      case 1: System.out.println("k lower than 2."); break;
      case 2: System.out.println("k equals 2."); break;
      case 3: System.out.println("k bigger than 2."); break;
      default: System.out.println("K = " + k);
    }
    return;
}
```

Example 4

Honeywell

```
private static void foo(int j){
    Integer k;
    switch(k){
      case 1: System.out.println("k lower than 2."); break;
      case 2: System.out.println("k equals 2."); break;
      case 3: System.out.println("k bigger than 2."); break;
      default: System.out.println("K = " + k);
    }
    return;
}
```

j is never used

1. j variable is never used and thus redundant

Example 4

Honeywell

```
private static void foo(int j){
    Integer k;
    switch(k){
      case 1: System.out.println("k lower than 2."); break;
      case 2: System.out.println("k equals 2."); break;
      case 3: System.out.println("k bigger than 2."); break;
      default: System.out.println("K = " + k);
    }
    return;
}
```

j is never used

k not initialized

1. j variable is never used and thus redundant
2. k variable is never initialized and thus unusable

Example 5

Honeywell

```
public void foo(){
  Item item = new Item();
  if(item.getInfo() != null){
    String info = item.getInfo().trim();
  }
}


class Item{
  public String getInfo(){
    // Making REST Request
  }
}
```

Example 5

Honeywell

```
public void foo(){
  Item item = new Item();
  if(item.getInfo() != null){
    String info = item.getInfo().trim();
  }
}
```

may return `null`

```
class Item{
  public String getInfo(){
    // Making REST Request
  }
}
```

REST may fail and return null

Static Code Analysis - Tools

Honeywell

- Klocwork
- SonarQube
- Findbugs
- Kiuwan
- Others
  - Compilers
  - IDEs
    - IntelliJ Idea
    - Eclipse

Capability Maturity Model Integration (CMMI)

Honeywell

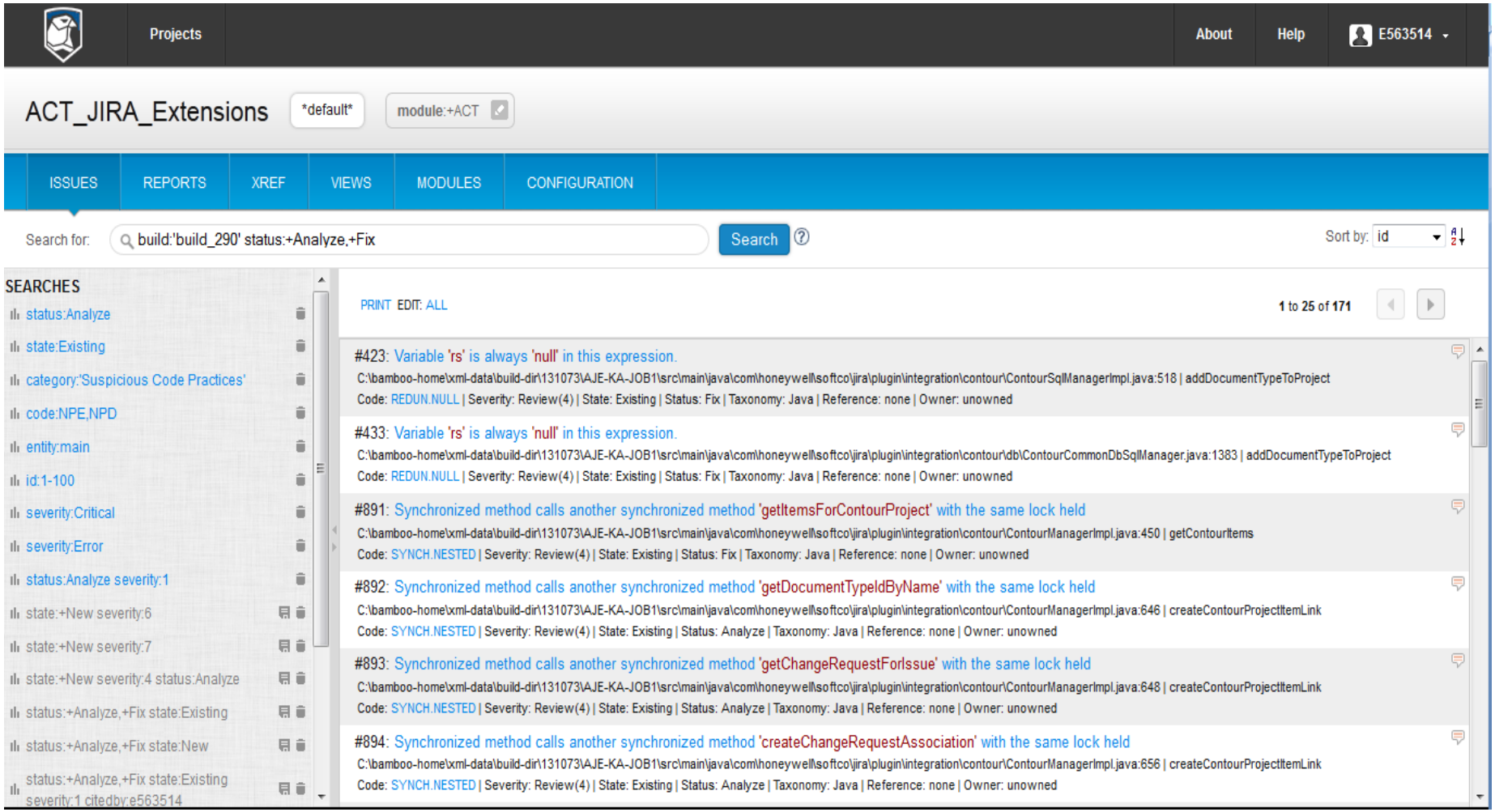
- **A set of rules defining the maturity of the company**
- **5 levels**
- **Honeywell achieving level 5**
  - continuous improvement of the processes and evaluation of the results across all the software development disciplines according to collected measurements and metrics
- **ACS Software Development Process (ASDP)**
  - Process used for Software Development across ACS
  - Aims to be compliant with CMMI level 5
  - Static Code Analysis is one of the steps in the Implementation discipline

Klocwork

- **Static Code Analysis tool**
- **Supported languages are C/C++, C#, Java**
- **Identifies code vulnerabilities**
  - Logical errors
  - Security vulnerabilities
  - Coding standards violations
- **Klocwork calculates software metrics such as lines of code, lines of comments, cyclomatic complexity, number of functions/methods**
- **Web Interface and user instance of Klocwork**
  - poor REST API
  - various restrictions on the side of Klocwork query language

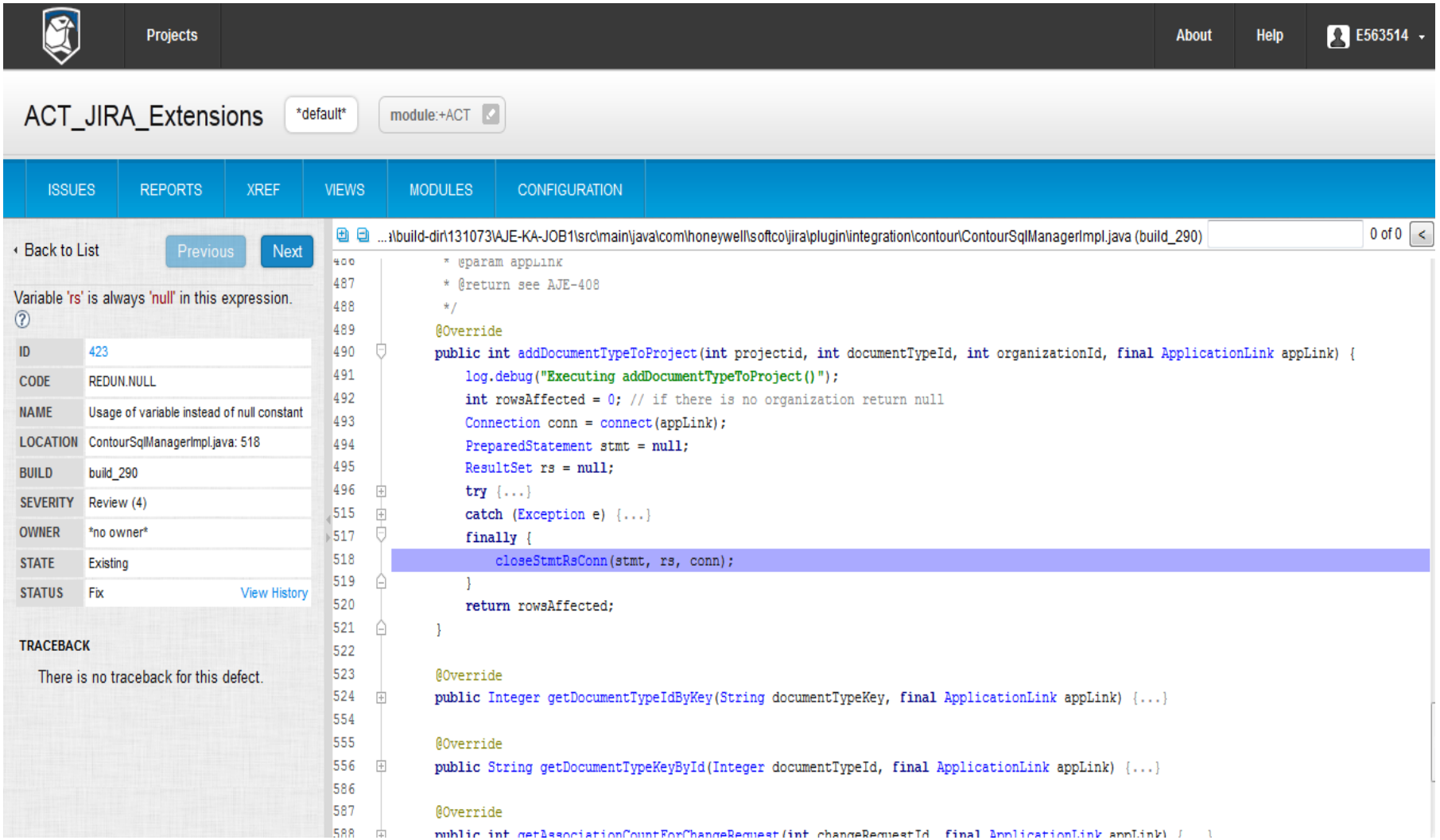
Klocwork

Honeywell



Klocwork

Honeywell



Tools Integrated with Klocwork

Honeywell

- **JIRA**
- **Crucible/FishEye**
- **Reasons**
  - Klockwork is not very user friendly
  - Results of SCA serve as input parameters for processes
    - These processes have majority of required data in other tools
  - Simplifying the processes
  - Delegation of control over SCA results

Defect Tracking Integration

Honeywell

Issue/Defect Tracking

Honeywell

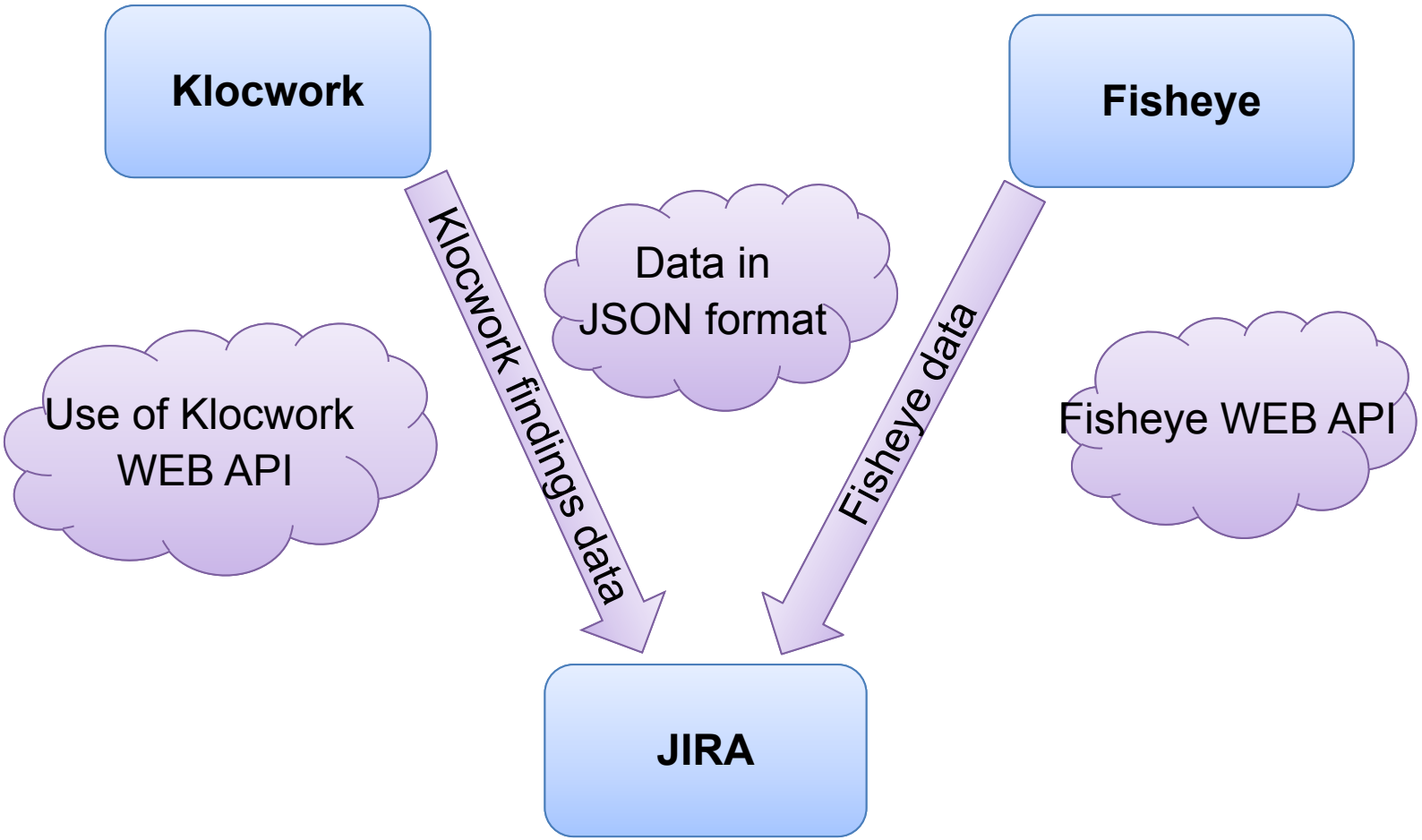
- **Honeywell ACS uses JIRA**
- **Issue and Project tracking software**
  - tracks time spent on resolving issues, progress of the project and various metrics connected to software development
  - one of the main tools used across Honeywell ACS
- **Issue**
  - a software bug (defect), a task, a helpdesk ticket, a new feature etc.
- **Easily extensible with wide variety of extensibility options**
- **Great integration capabilities with other Atlassian products**

Source Code Management

- Fisheye
- Web Interface for read only access to SVN
- Visualization and reporting capabilities regarding source code
- Searching capabilities according to commits, comments, people etc.
- Great WEB API for communication between tools

JIRA – Klocwork – Fisheye Integration Overview

Honeywell



Goals

- Development cycle time reduction
- Code quality improvement
- Build basis for creating new more sophisticated metrics combining defect/issue tracking and static code analysis
- Set basis for future development of integration and possible defect prediction

Create JIRA defect from Klocwork

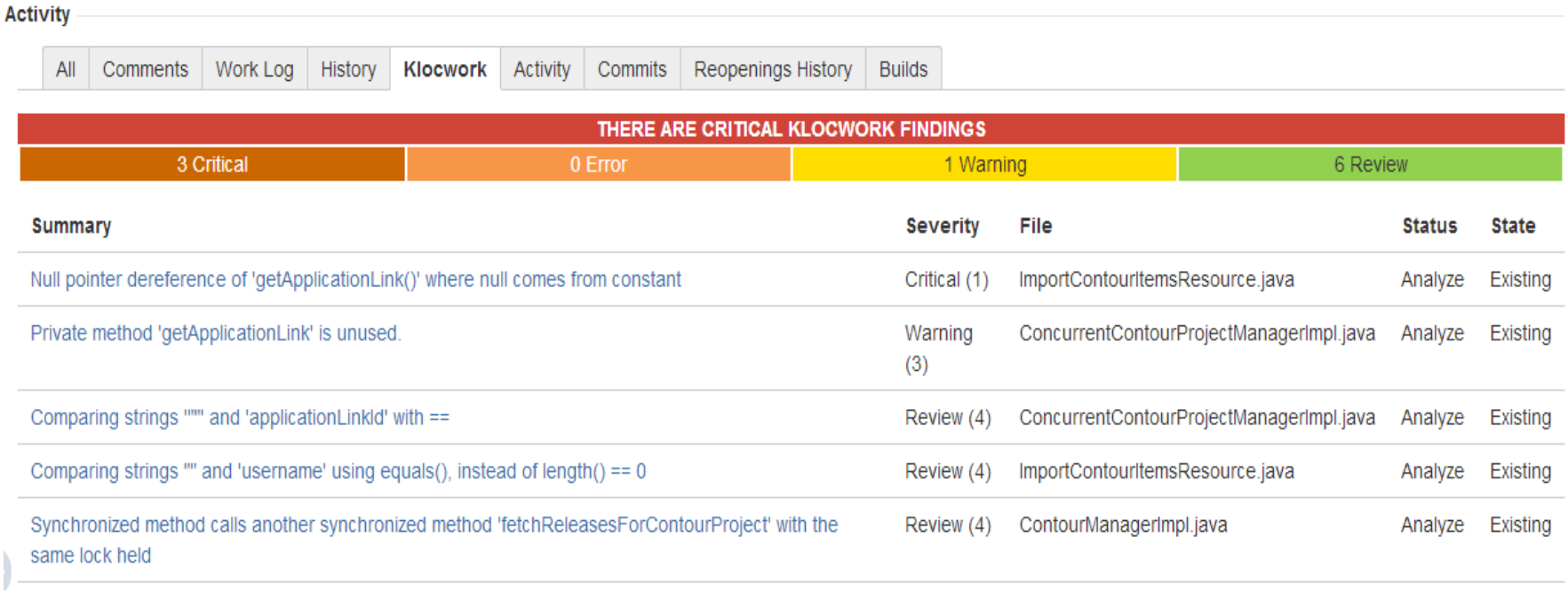
Honeywell

- A simple button in Klocwork Finding
- Creates a JIRA defect with as much information as possible
- Stores the data about defect resolving in JIRA
- Not able to get line numbers
- Not able to get specific faulty code

Create a list of Klocwork findings for specific JIRA issue

Honeywell

- Integrates data from Fisheye with data from Klocwork and creates a list of Klocwork Findings that were introduced into the code as part of implementation of some Issue
- Higher Code Quality
- Lower Development Cycle Time
- Information about the quality of the feature



Report

Honeywell

- Indicates whether the code is ready for testing and release
- Gives overall status about the code quality written as part of the selected version of the product

Version Release Readiness Report

Version Analyzed	Release 4.1
Latest Klocwork Analysis	build_265 - 26/Mar/14 2:06 AM

THERE ARE UNRESOLVED KLOCWORK FINDINGS WITH CRITICAL OR ERROR SEVERITY

	Critical	Error	Warning	Review
AJE-1266	0	0	0	1
AJE-1145	3	0	1	6
AJE-1144	10	0	1	17
AJE-1263	0	0	0	1
Summary	13	0	2	25

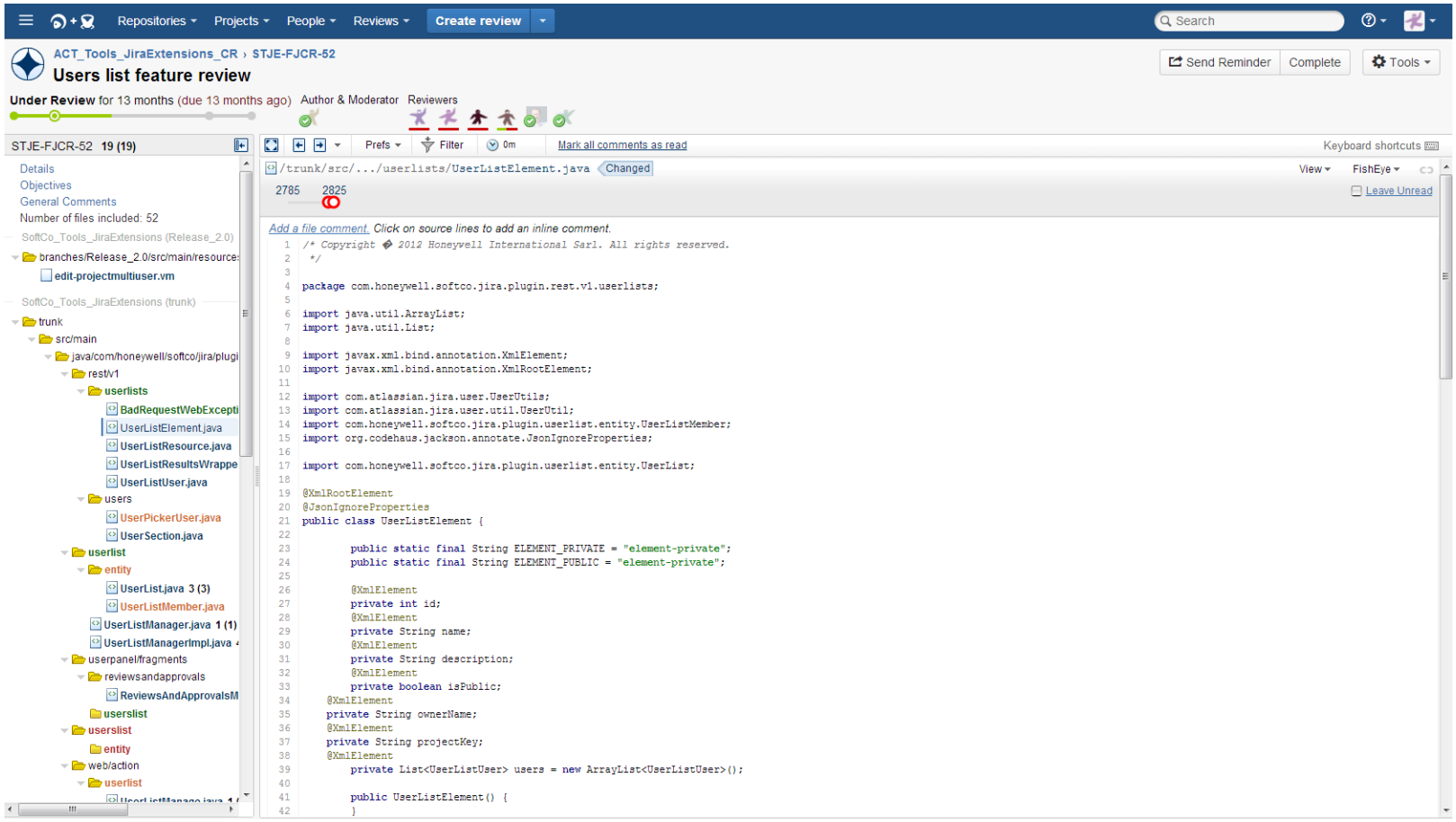
Manual Code Reviews Integration

Honeywell

Manual Code Reviews

Honeywell

- Systematic examination of the source code
- Used to verify the code from various perspectives
- Atlassian Crucible

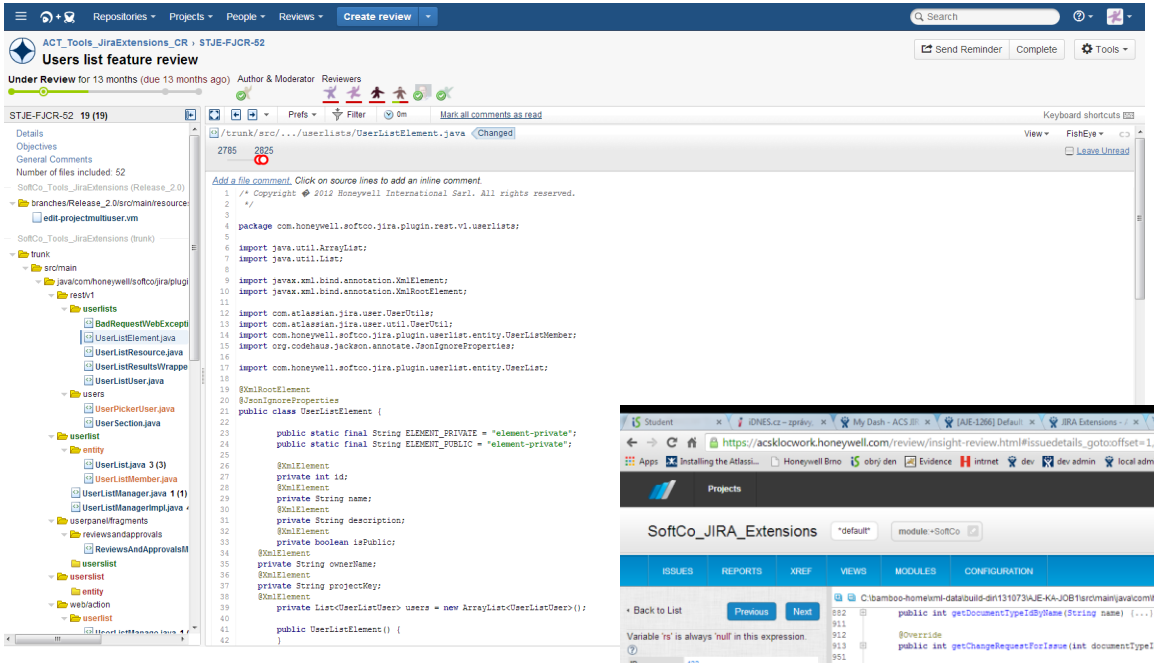


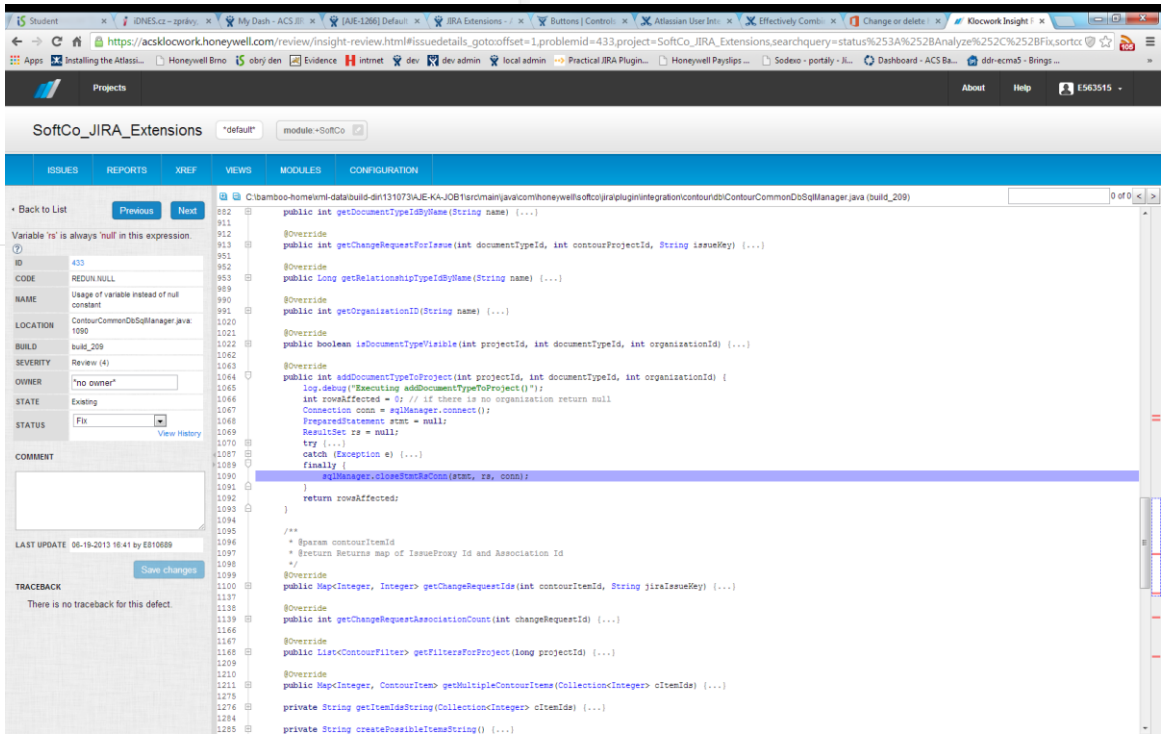
Testing Readiness

- Simple rule: cost of a defect rises with the time it is not discovered
  - Apply even for defects found during testing activities
- Solution: try to find defects as soon as possible
  - Apply all available tools/processes
  - SCA
  - Manual Code Reviews
- Not popular activity
  - Need to use two tools for very similar activities
  - Inspecting more complex code can be confusing
  - Human Factor
  - Klocwork User Friendliness

How to improve?

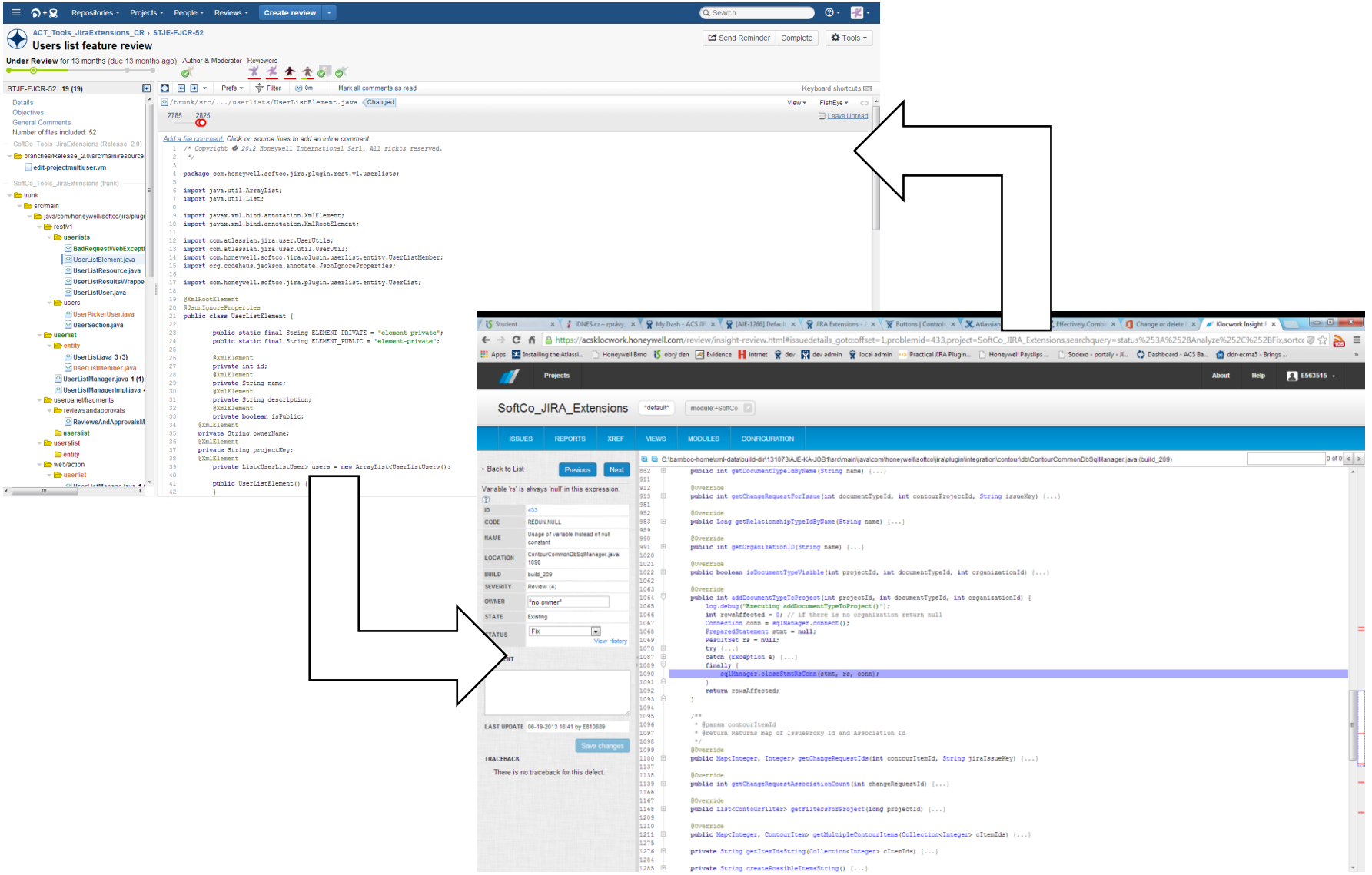
Honeywell





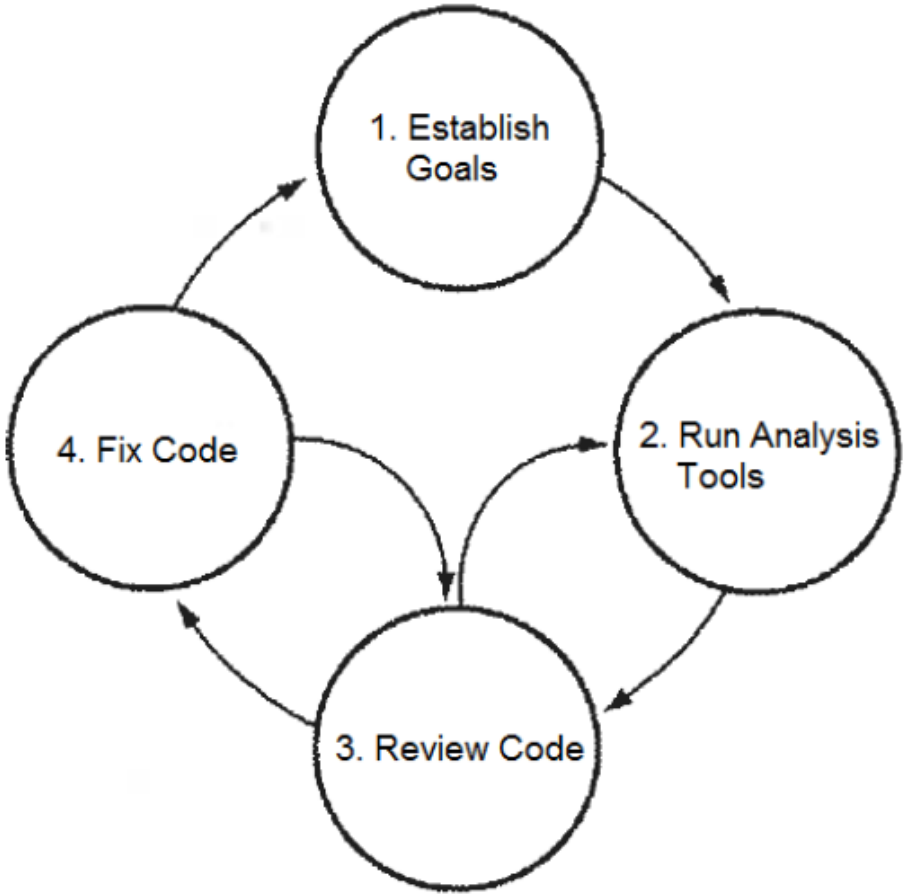
How to improve?

Honeywell



Integration

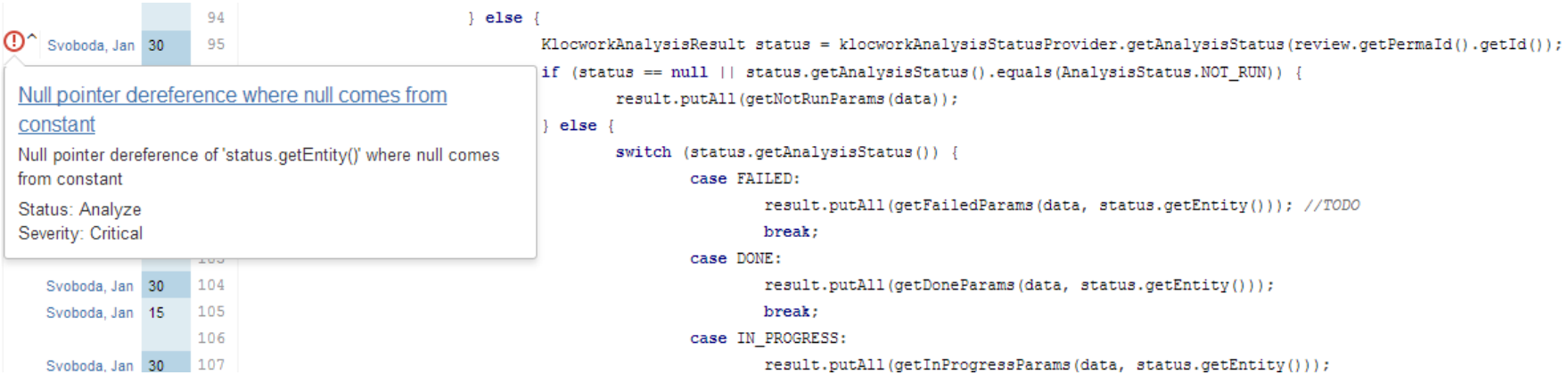
- Both of the tools looks similar
  - Why not integrate them?
- Benefits:
  - Time saving
  - User Friendliness
  - Process Enforcement
- Features
  - Source File View
  - File Tags
  - Analysis Overview panel
  - Hot Spot Review



Source File View

Honeywell

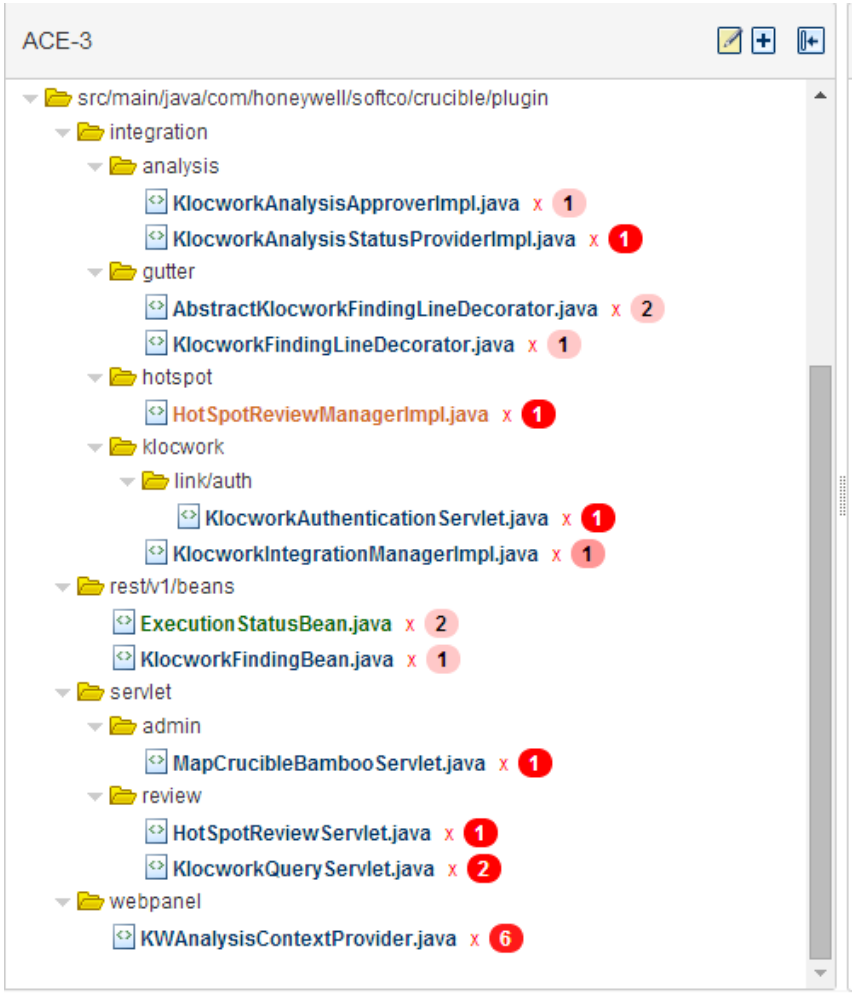
- Displays Klocwork findings directly in Crucible source code view



File Tags

Honeywell

- Differentiate files under review based on the number and severity of Klocwork findings



Analysis Overview panel

Honeywell

- Displays statistic data about Klocwork findings in the review

Hot Spot Review

Honeywell

- „Hot Spot“ = parts of the code satisfying some condition

Thank You

Honeywell

Q&A

Honeywell

www.honeywell.com