Summarization Techniques for Code, Change, Testing and User Feedback

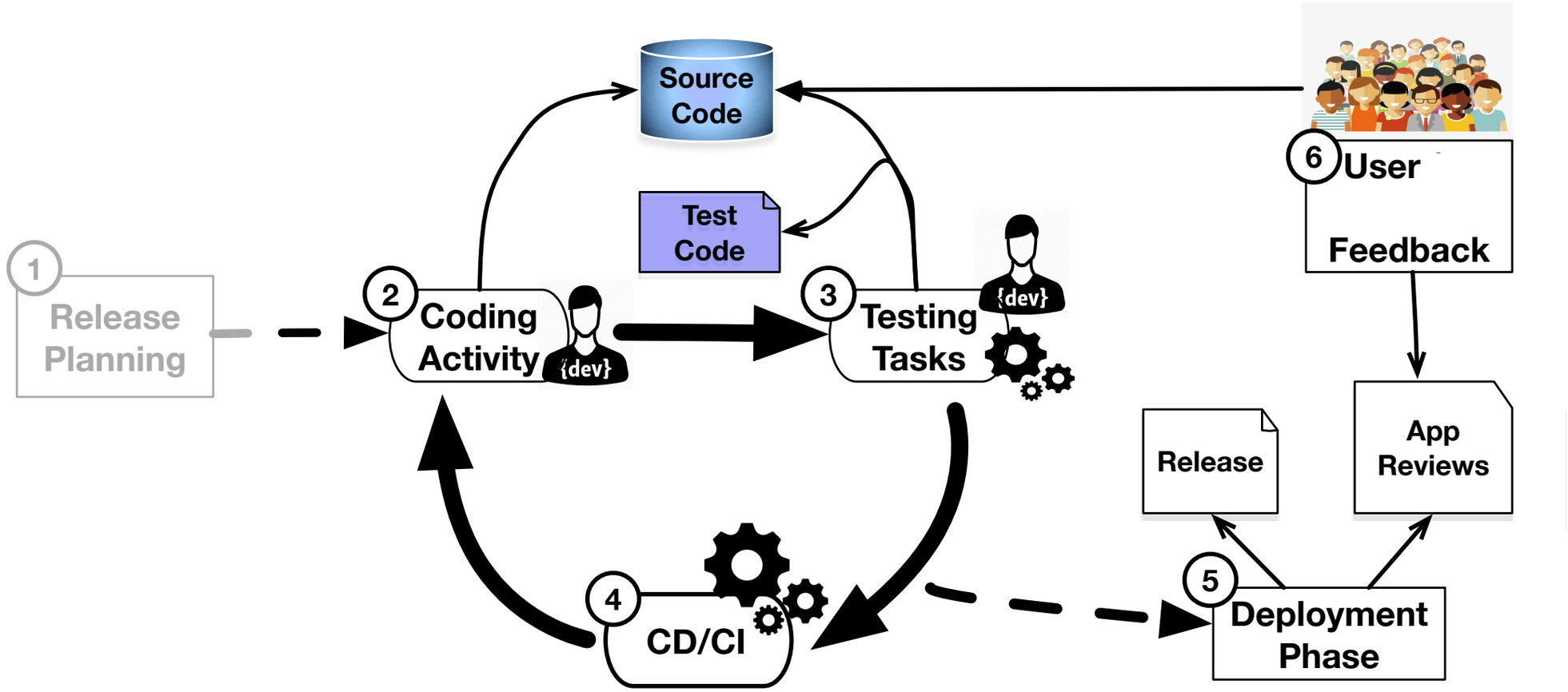
Vienna Software Seminar (2017)

Sebastiano Panichella
Institut für Informatik
Universität Zürich
panichella@ifi.uzh.ch

University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Release Cycle



Release Cycle

Performance Testing

Microservices

CD/CI

Automated testing

Coding activities

Maintenance activities

Code Review

Manual testing

A/B testing

Structured e-mails

Survey

App reviews

Security

…

Team coordination

Software documentation

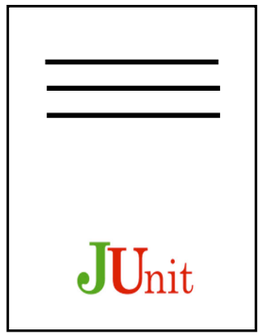
…

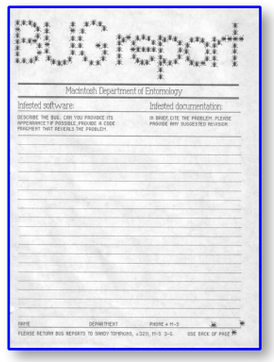
Crowdsourcing

…

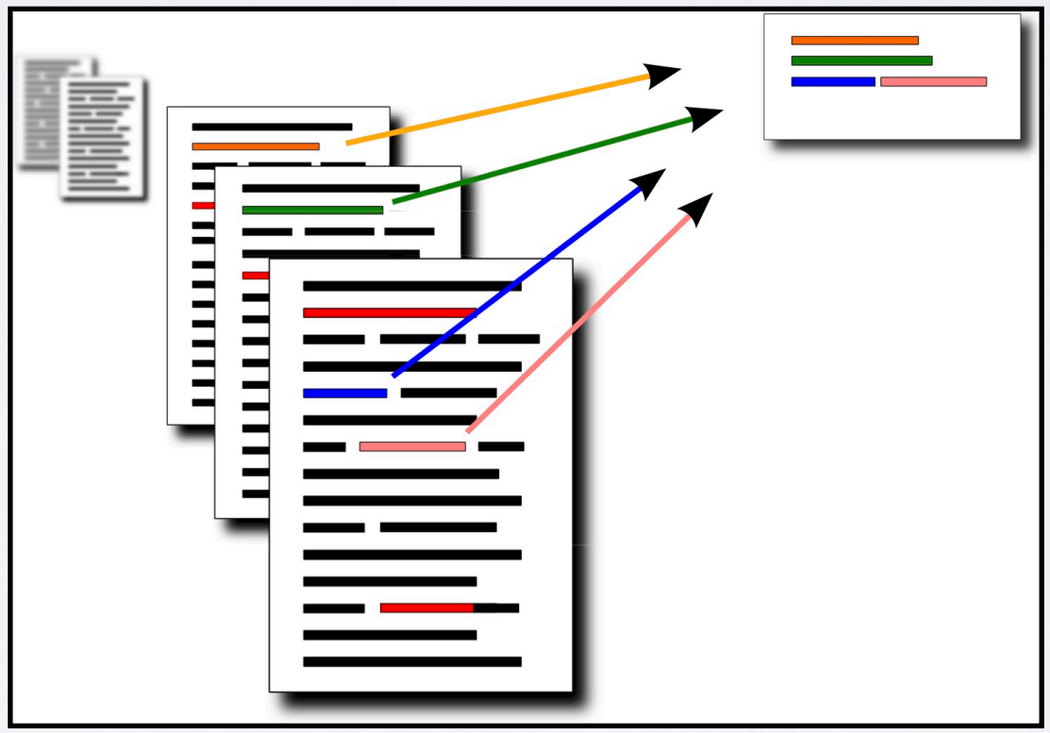
Release Cycle

"*In modern software companies it is nowadays,* crucial to enact a software development process able to dynamically react to **market requirements** (i.e., users requests), delivering at same time *high* ***quality and reliable software***".





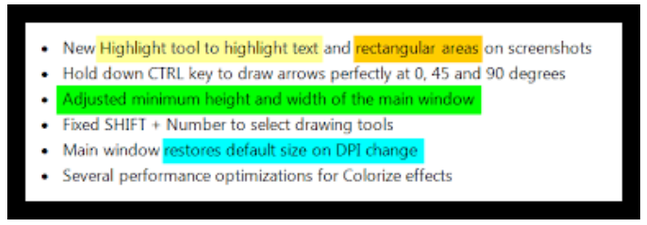
Summarization Approaches



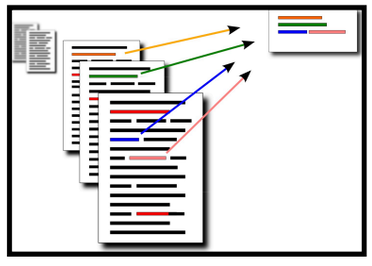
Summarization Approaches

"…*have the general capability of automatically extracting or abstracting key content from one or more sources of information thus, determining the relevant information in the source being summarized and reducing its content…*"

1) indicative summary



2) informative summary



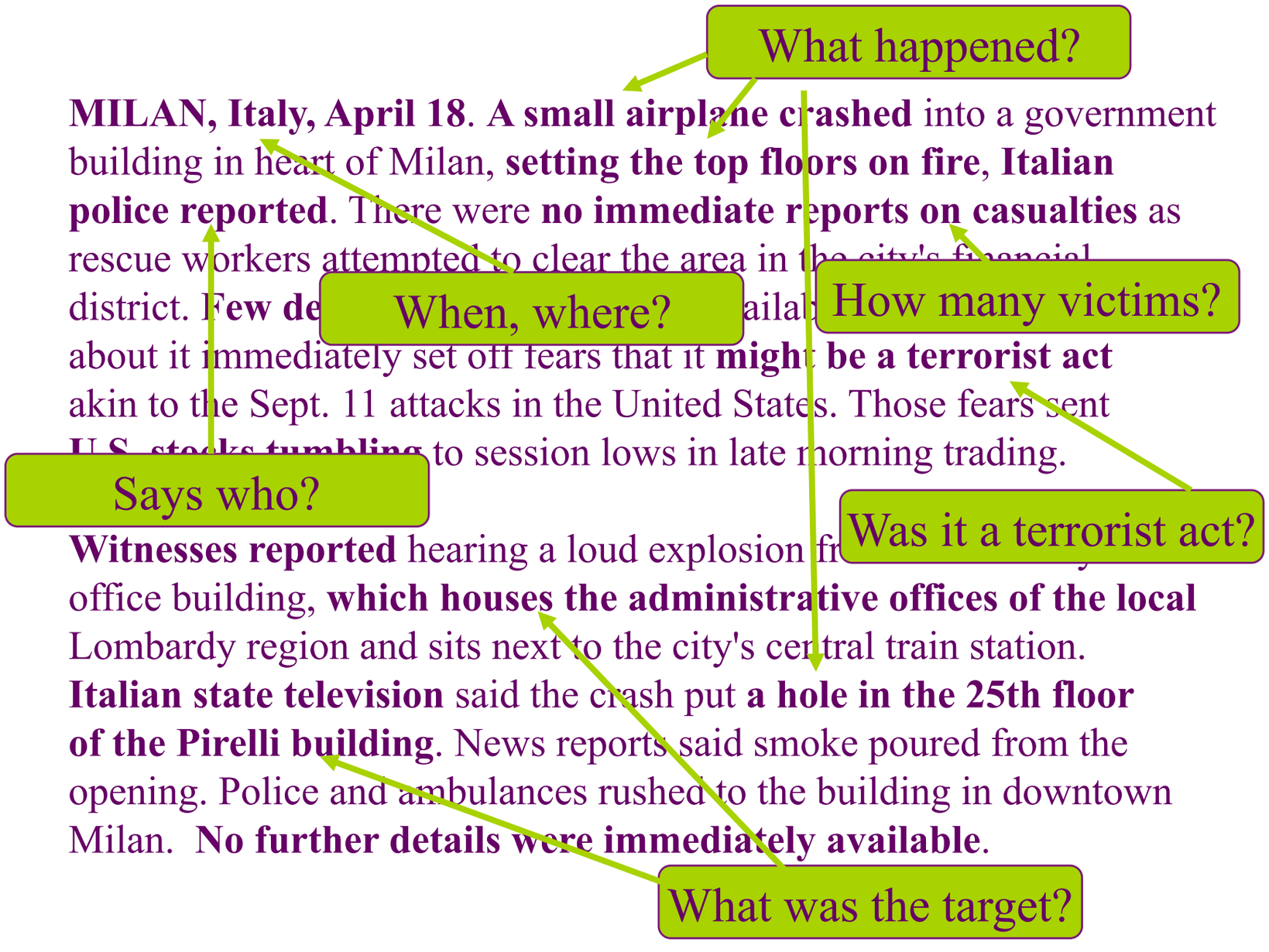
3) critical summary (or review)

Example of informative summary

MILAN, Italy, April 18. **A small airplane crashed** into a government building in heart of Milan, **setting the top floors on fire**, **Italian police reported**. There were **no immediate reports on casualties** as rescue workers attempted to clear the area in the city's financial district. **Few details of the crash** were available, but news reports about it immediately set off fears that it **might be a terrorist act** akin to the Sept. 11 attacks in the United States. Those fears sent **U.S. stocks tumbling** to session lows in late morning trading.

Witnesses reported hearing a loud explosion from the 30-story office building, **which houses the administrative offices of the local** Lombardy region and sits next to the city's central train station. **Italian state television** said the crash put **a hole in the 25th floor of the Pirelli building**. News reports said smoke poured from the opening. Police and ambulances rushed to the building in downtown Milan. **No further details were immediately available**.

Example of informative summary



MILAN, Italy, April 18. **A small airplane crashed** into a government building in heart of Milan, **setting the top floors on fire**, **Italian police reported**. There were **no immediate reports on casualties** as rescue workers attempted to clear the area in the city's financial district. **Few de** ailab about it immediately set off fears that it **might be a terrorist act** akin to the Sept. 11 attacks in the United States. Those fears sent **U.S. stocks tumbling** to session lows in late morning trading.

Witnesses reported hearing a loud explosion fr office building, **which houses the administrative offices of the local** Lombardy region and sits next to the city's central train station. **Italian state television** said the crash put **a hole in the 25th floor of the Pirelli building**. News reports said smoke poured from the opening. Police and ambulances rushed to the building in downtown Milan. **No further details were immediately available**.

Example of informative summary

Text Compactor

Free Online Automatic Text Summarization Tool

Home

About

Follow these simple steps to create a summary of your text.

Step 1

Type or paste your text into the box.

Albert Einstein was born on March 14, 1879 in Ulm, the first child of the Jewish couple Hermann and Pauline Einstein, née Koch. In June 1880 the family moved to Munich where Hermann Einstein and his brother Jakob founded the electrical engineering company Einstein & Cie. Albert Einstein's sister Maria, called Maja, was born on November 18, 1881. Einstein's childhood was a normal one, except that to his family's irritation, he learnt to speak at a late age. Beginning in 1884 he received private education in order to get prepared for school. 1885 he started learning to play violin. Beginning in 1885 he received his primary education at a Catholic school in Munich (Petersschule); in 1888 he changed over to the Luitpold-Gymnasium, also in Munich. However, as this education was not to his liking and, in addition, he did not get along with his form-master he left this school in 1894 without a degree and joined his family in Italy where they had settled meanwhile.
In order to be admitted to study at the "Eidgenoessische Polytechnische Schule" (later renamed ETH) in Zurich, Einstein took his entrance examination in October 1895. However, some of his results were insufficient and, following the advice of the rector, he attended the "Kantonsschule" in the town of Aarau in order to improve his knowledge. In early October 1896 he received his school-leaving certificate and shortly thereafter enrolled at the Eidgenoessische Polytechnische Schule with the goal of becoming a teacher in Mathematics and Physics. Einstein, being an average student, finished his studies with a diploma degree in July 1900. He then applied, without success, for assistantships at the Polytechnische Schule and other universities. Meanwhile he had abandoned the German citizenship and formally applied for the Swiss one which he was granted on February 21, 1901.

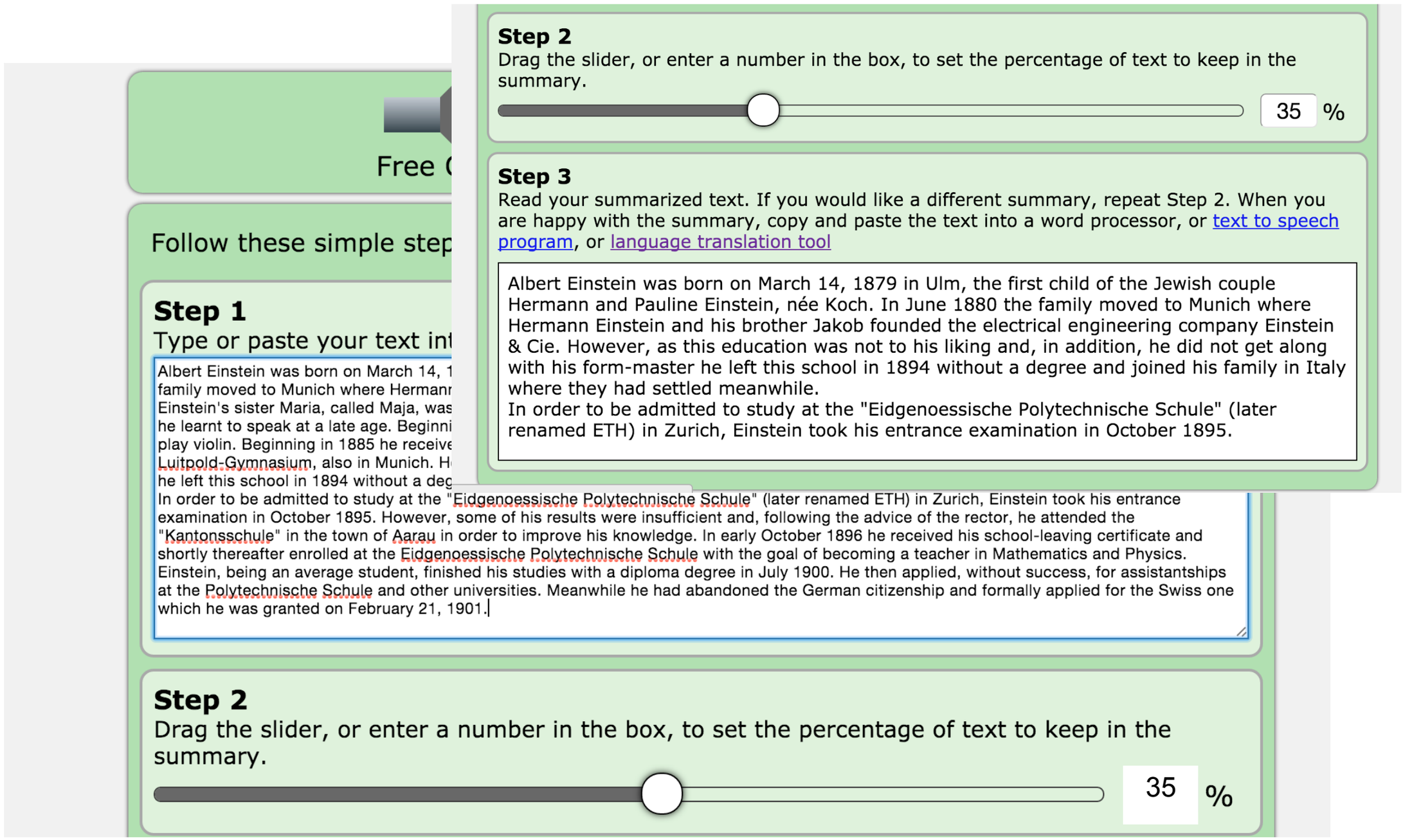
Step 2

Drag the slider, or enter a number in the box, to set the percentage of text to keep in the summary.

35 %

http://textcompactor.com/

Example of informative summary



http://textcompactor.com/

Release Cycle

Performance Testing

Microservices

CD/CI

Automated testing

Coding activities

Maintenance activities

Code Review

Testing

A/B testing

Structured e-mails

Survey

App reviews

?

Security

…

…

Team coordination

Software documentation

…

Crowdsourcing

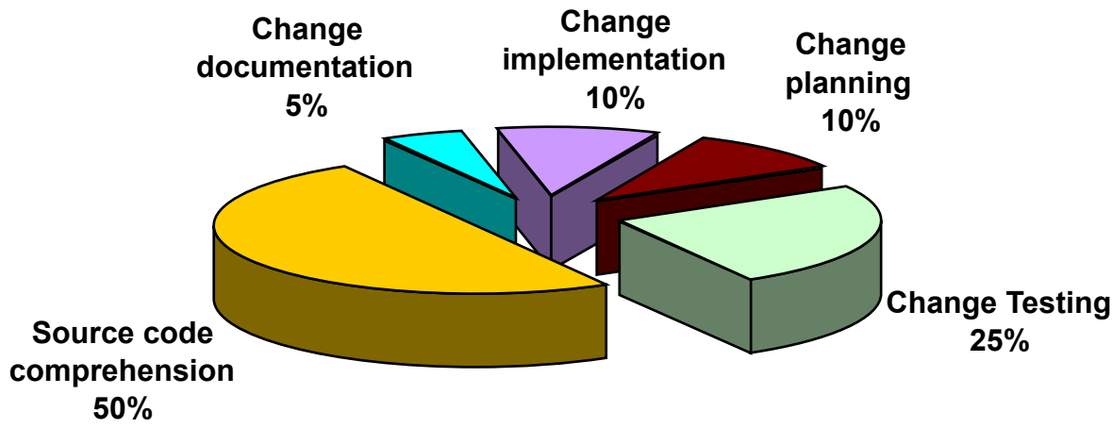
…

…

Release Cycle

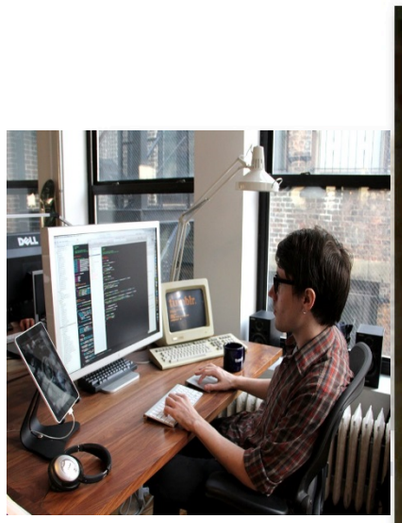
Performance Testing	**Coding activities**	**A/B testing**
Microservices	**Maintenance activities**	**Structured e-mails**
CD/CI	**Code Review**	**Survey**
Automated testing	**Testing**	**App reviews**
Security	**Team coordination**	**Crowdsourcing**
…	**Software documentation**	**…**
…	**…**	**…**

Source Code Summaries: Why?



Activities in Software Maintenance

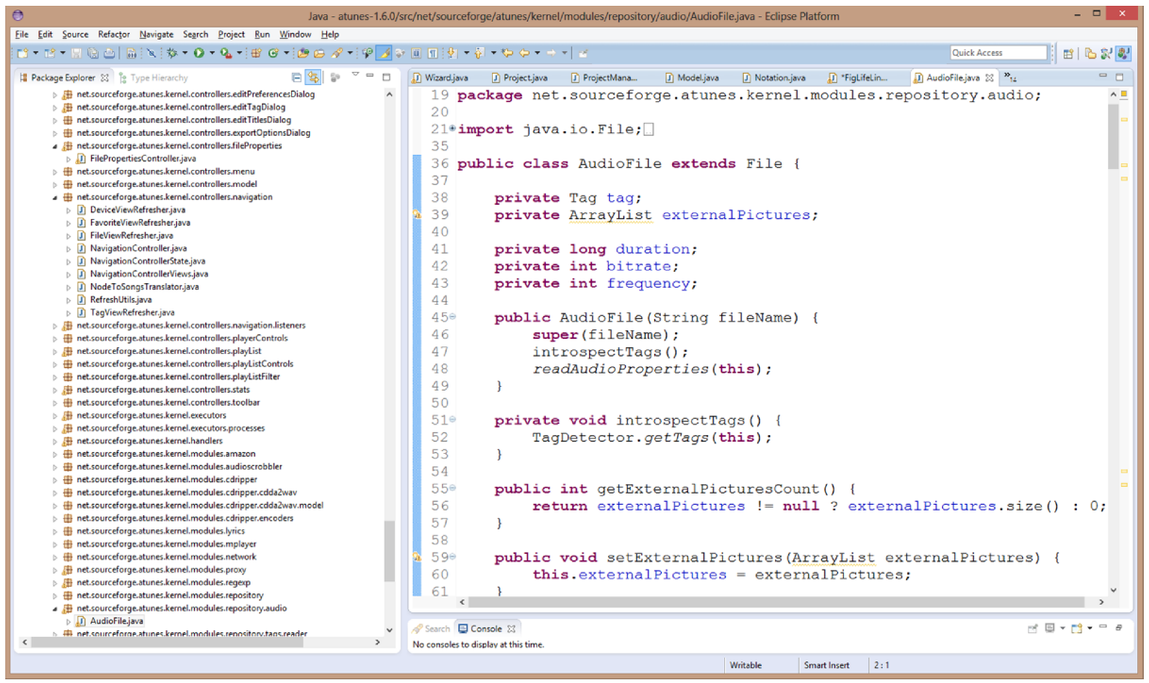
Source Code Summaries: Why?



Developers spend more time reading than writing code

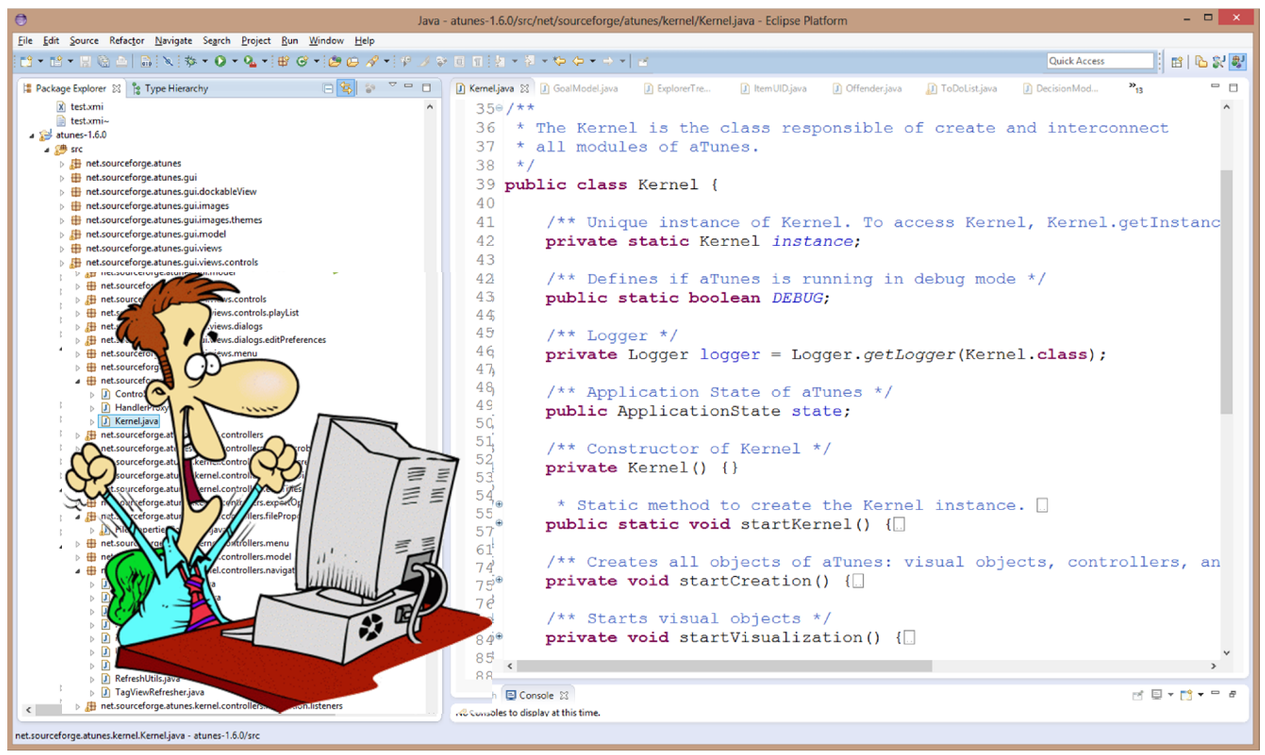
Understanding Code…

Not So Happy Developers



Absence of Comments in the Code

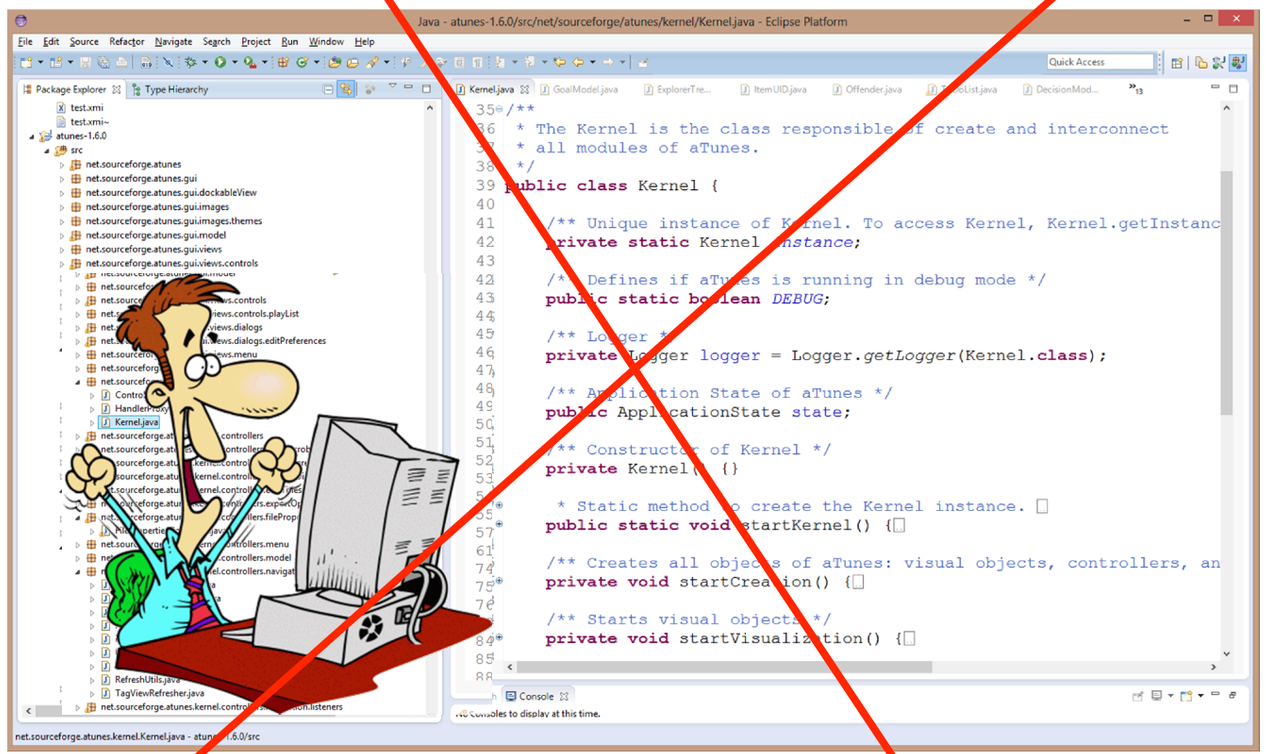
Happy Developers



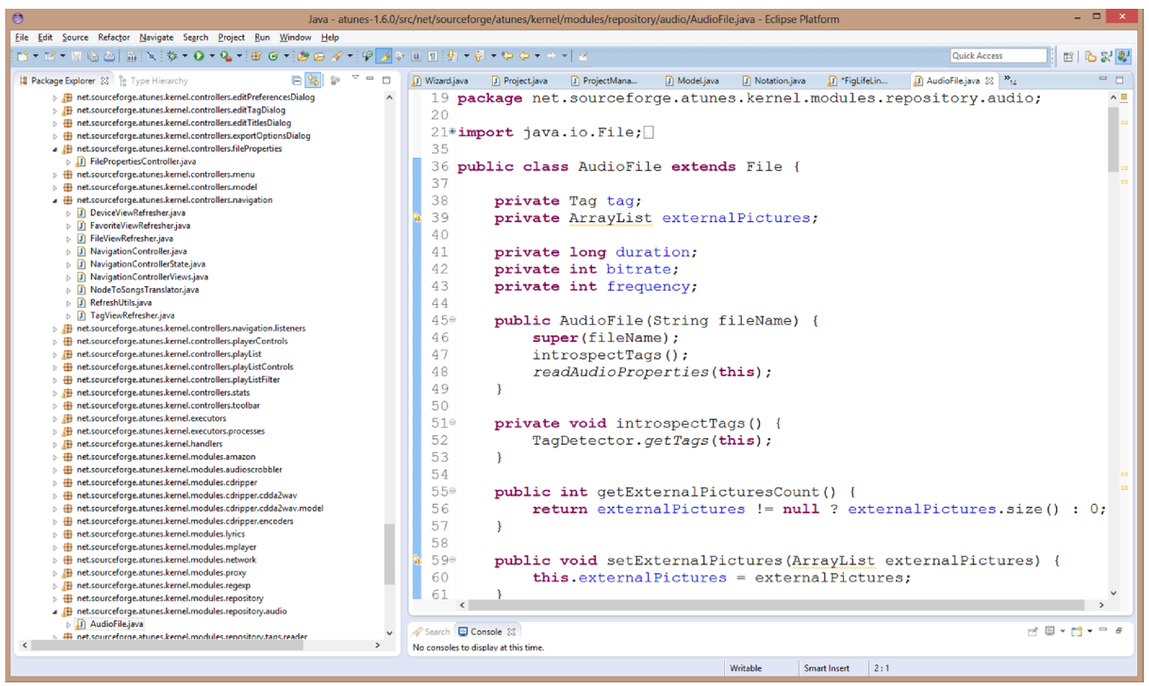
Comments in the Code

Understanding Code…

Happy Developers



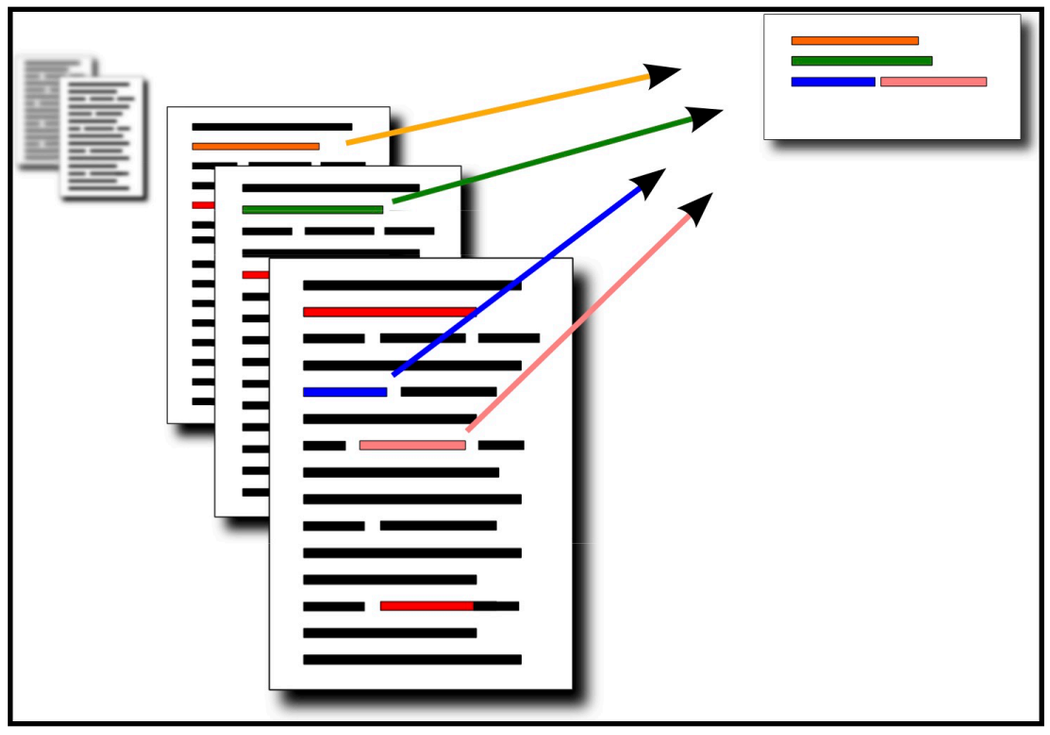
Not So Happy Developers



Absence of Comments in the Code

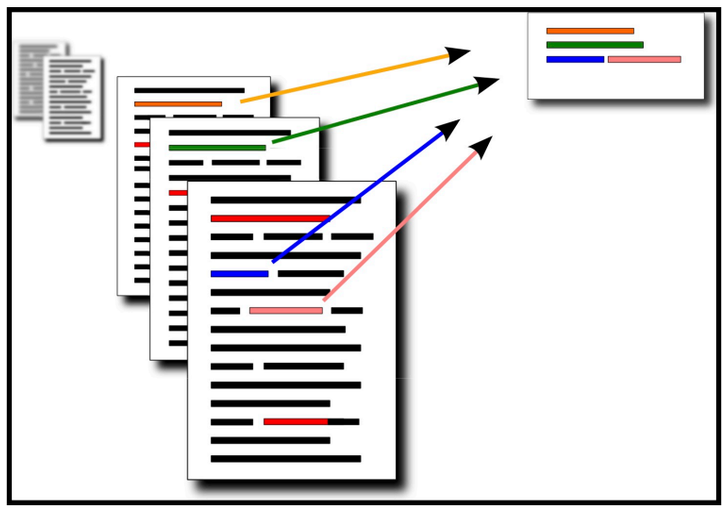
Comments in the Code

Source Code Summaries: How?



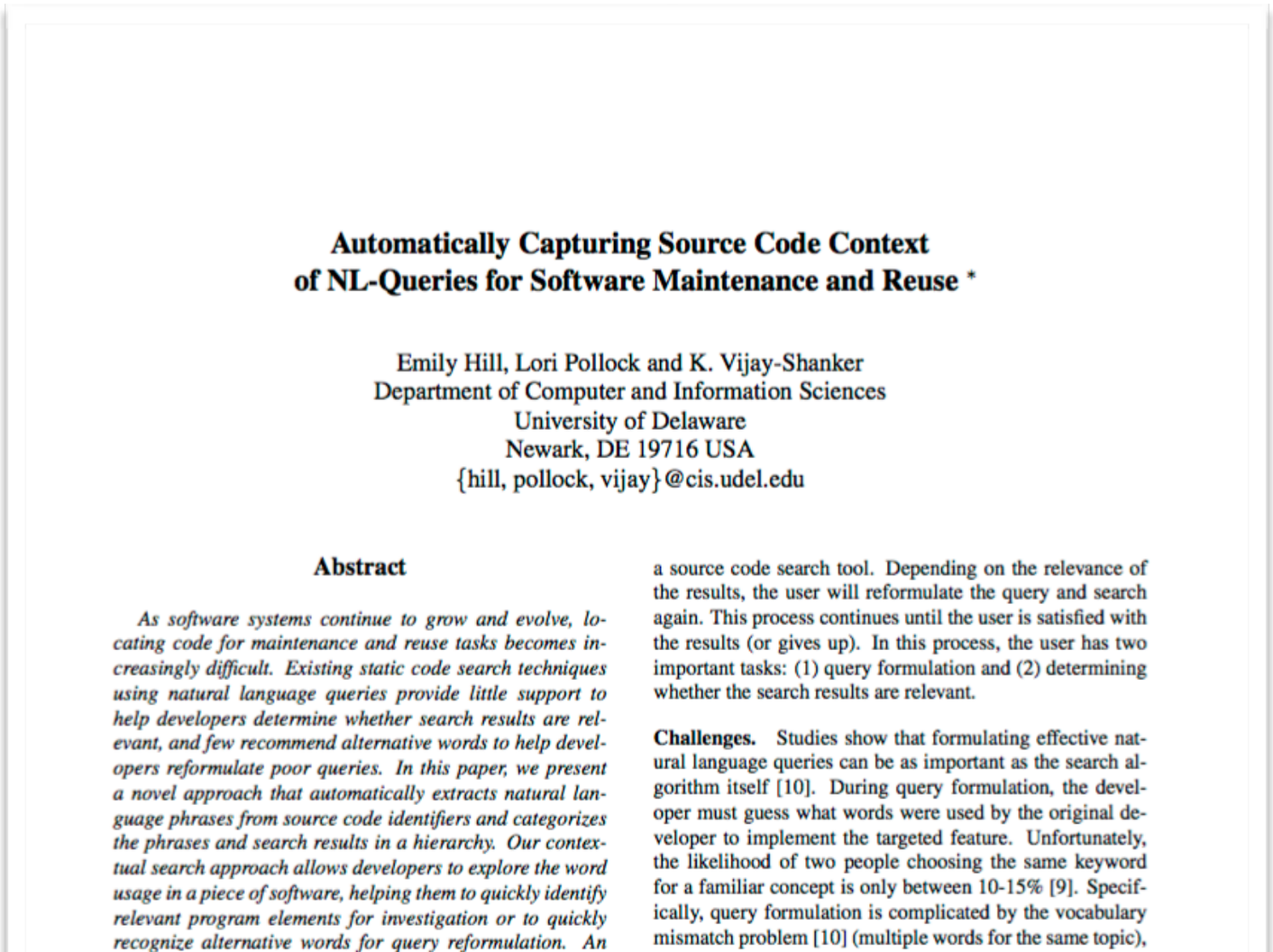
Questions when Generating Summaries of Java Classes

- 1) **What** information to include in the summaries?
- 2) **How to** generate and **present** the summaries?



Summary Generator

Software Words Usage Model: deriving *<actions>*, *<themes>*, and *<secondary arguments>* from class, methods, attributes and variable identifiers



Automatically Capturing Source Code Context of NL-Queries for Software Maintenance and Reuse *

Emily Hill, Lori Pollock and K. Vijay-Shanker
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716 USA
{hill, pollock, vijay}@cis.udel.edu

Abstract

As software systems continue to grow and evolve, locating code for maintenance and reuse tasks becomes increasingly difficult. Existing static code search techniques using natural language queries provide little support to help developers determine whether search results are relevant, and few recommend alternative words to help developers reformulate poor queries. In this paper, we present a novel approach that automatically extracts natural language phrases from source code identifiers and categorizes the phrases and search results in a hierarchy. Our contextual search approach allows developers to explore the word usage in a piece of software, helping them to quickly identify relevant program elements for investigation or to quickly recognize alternative words for query reformulation. An

a source code search tool. Depending on the relevance of the results, the user will reformulate the query and search again. This process continues until the user is satisfied with the results (or gives up). In this process, the user has two important tasks: (1) query formulation and (2) determining whether the search results are relevant.

Challenges. Studies show that formulating effective natural language queries can be as important as the search algorithm itself [10]. During query formulation, the developer must guess what words were used by the original developer to implement the targeted feature. Unfortunately, the likelihood of two people choosing the same keyword for a familiar concept is only between 10-15% [9]. Specifically, query formulation is complicated by the vocabulary mismatch problem [10] (multiple words for the same topic),

E. Hill et al. Automatically capturing source code context of NL-queries for software maintenance and reuse. ICSE 2009

Source Code Summaries: How?

When Navigating Java Classes…

```
package net.sourceforge.atunes.kernel.modules.repository.audio;

import java.io.File;

public final class AudioFile implements AudioObject, Serializable, Comparable<AudioFile> {

    private static final long serialVersionUID = -1139001443603556703L;

    private static transient Logger logger = new Logger();

    private File file;
    protected Tag tag;
    private List<File> externalPictures;
    protected long duration;
    protected long bitrate;
    protected int frequency;
    protected long readTime;
    private int stars = 0;

    public AudioFile(String fileName) {
        readFile(new File(fileName));
    }

    private void readFile(File file) {
        this.file = file;

        if (!isApeFile(file) && !isMPCFile(file)) {
            introspectTags();
            readAudioProperties(this);
        }
        this.readTime = System.currentTimeMillis();
    }
```

we look at

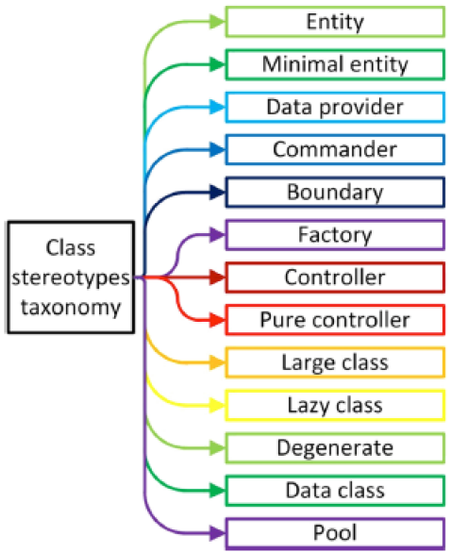
- *Name of the Class*
- *Attributes*
- *Methods*
- *Dependencies between Classes*

https://github.com/larsb/atunesplus/blob/master/aTunes/src/main/java/net/sourceforge/atunes/kernel/modules/repository/audio/AudioFile.java

Source Code Summaries: How?

When Navigating Java Classes…

- Generic responsibilities (domain independent)
  - Class stereotypes [Dragan et al., ICSM'10]
  - E.g., data class, entity, controller, boundary, et



Source Code Summaries: How?

When Navigating Java Classes…

- Generic responsibilities (domain independent)
  - Class stereotypes [Dragan et al., ICSM'10]
  - E.g., data class, entity, controller, boundary, etc.

Entity

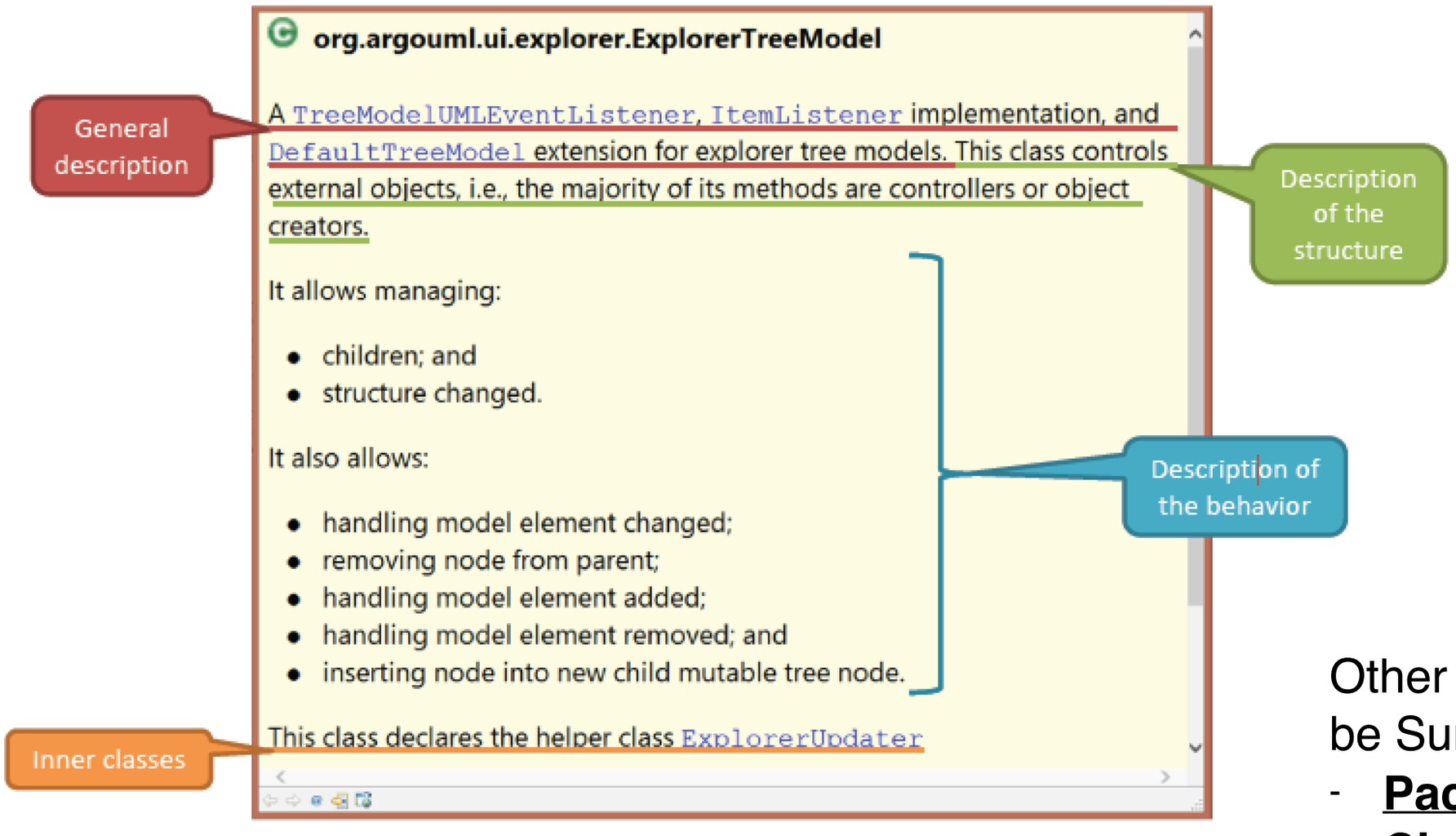
Class stereotypes taxonomy

```
public class CdRipper {

    private Cdda2wav cddawav;
    private Encoder encoder;
    private ProgressListener listener;

    private boolean interrupted;

    private String artist;
    private String album;

    private String fileNamePattern;

    public static final String ARTIST_PATTERN = "%A";
    public static final String ALBUM_PATTERN = "%L";
    public static final String TRACK_TITLE_AND_NUMBER = "%TN";

    public CdRipper() {

    public CDInfo getCDInfo() {

    public boolean ripTracks(ArrayList<Integer> tracks,

    private String getFileName(ArrayList<String> titles,
```

Keeper of the data model and/or business logic

[Dragan et al., ICSM'10]
[Moreno et al., ASE'12]

How to present and generate the summaries?

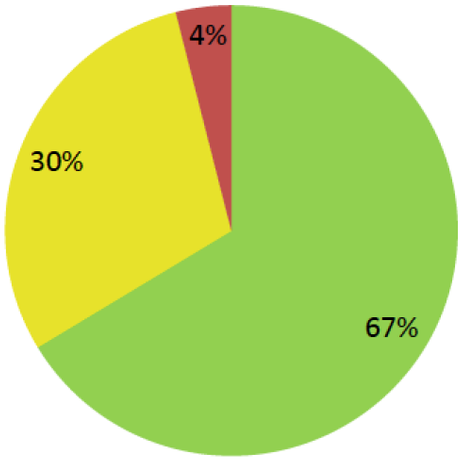


http://www.cs.wayne.edu/~severe/jsummarizer/

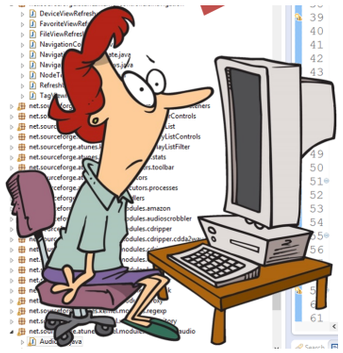
Other Code Artefacts can be Summarised as well:

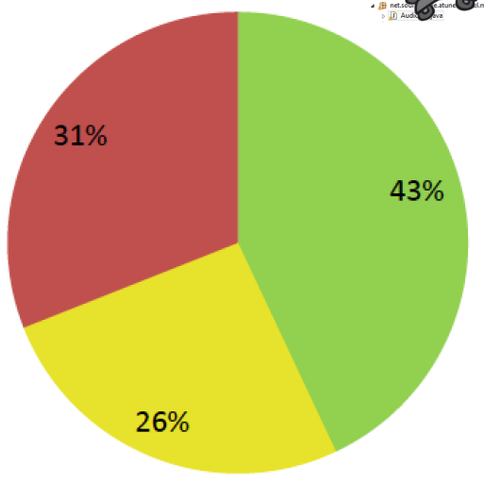
- **Packages**
- **Classes**
- **Methods**
- **etc.**

Evaluation of the Summaries



- Is easy to read and understand
- Is somewhat readable and understandable
- Is hard to read and understand

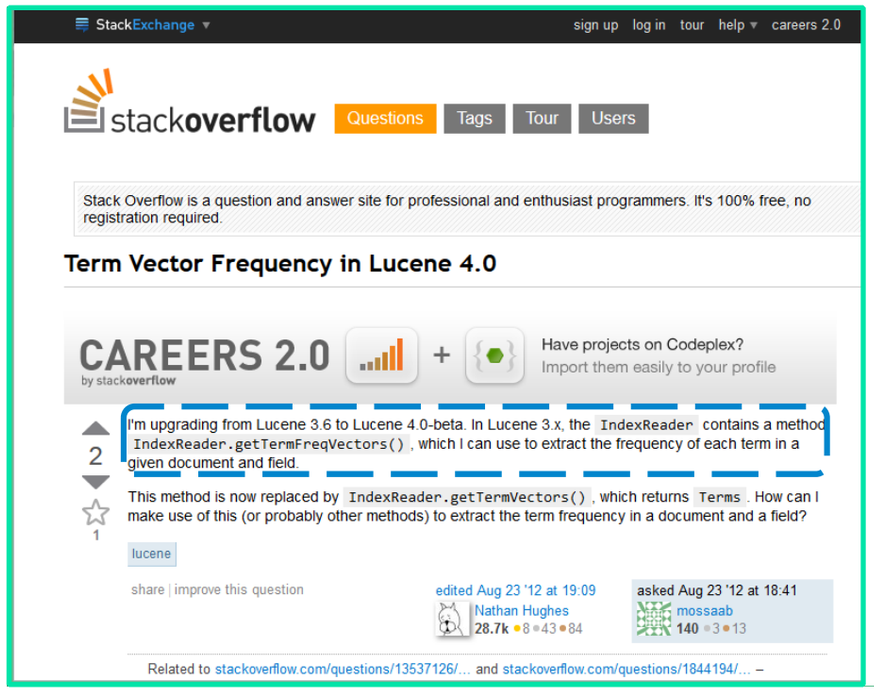


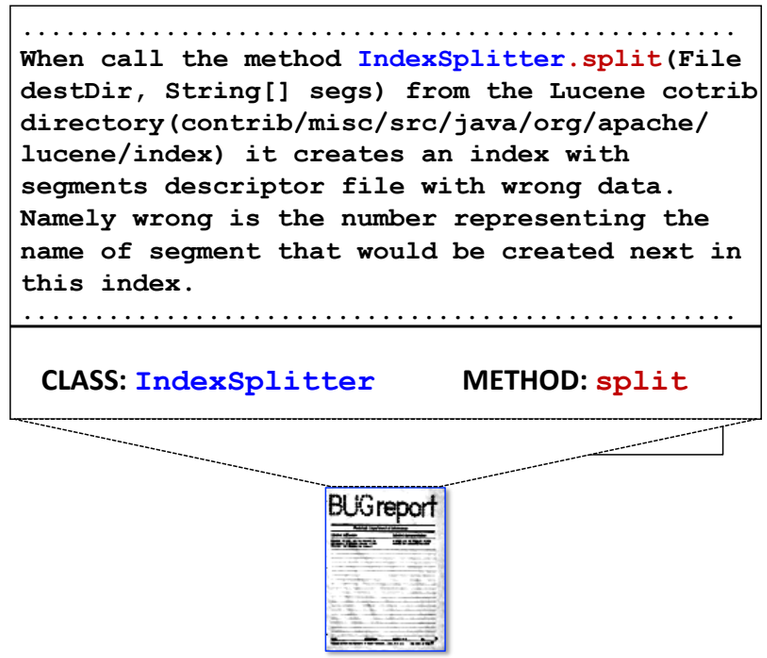


- Not missing any information
- Missing some information
- Missing some very important information

[Moreno at al. - ICPC 2013]

Potential Useful Code Descriptions can be Found in Developers' Discussions…





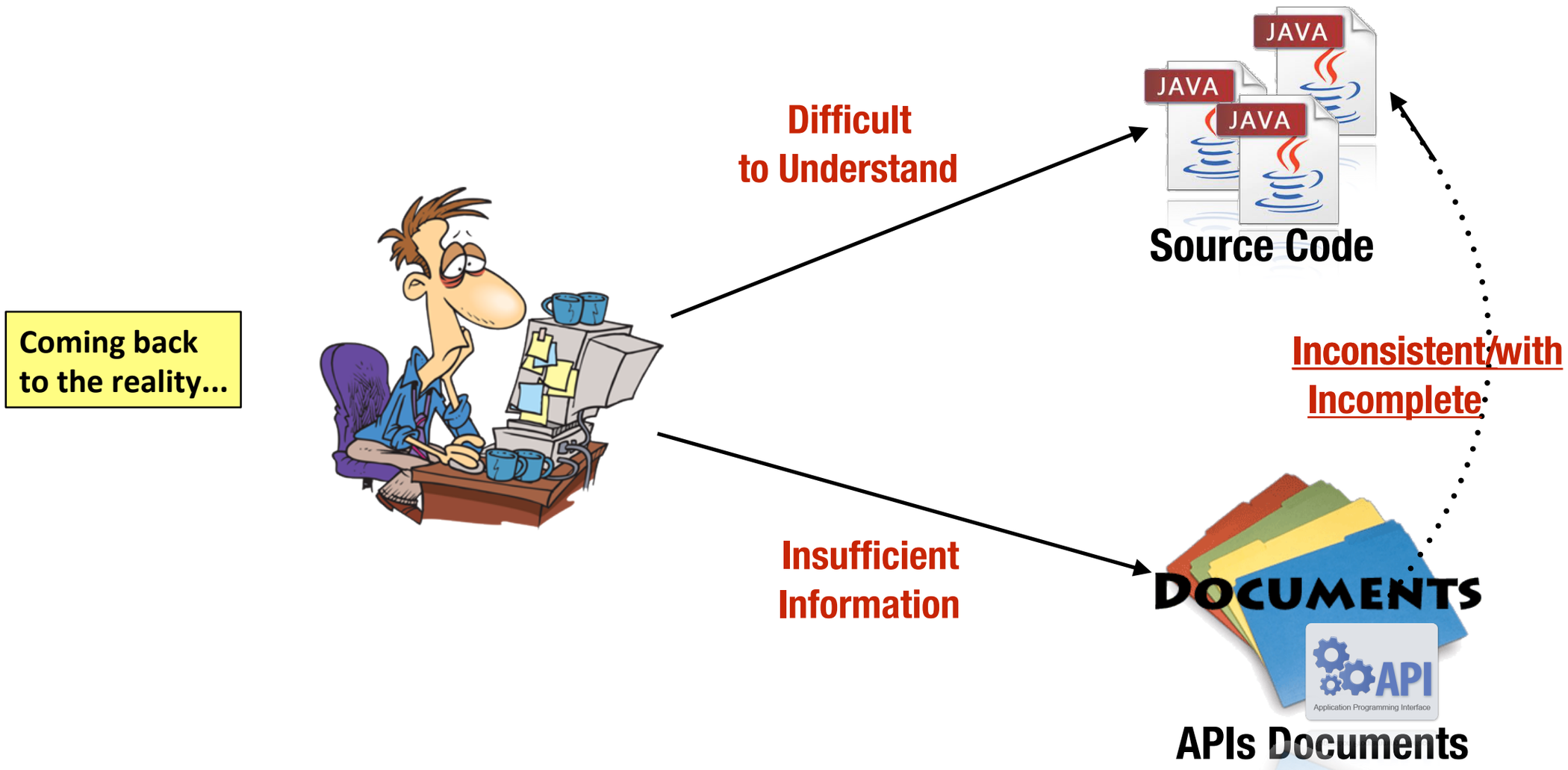
- Others researchers proposed to detect source code descriptions from sources external to the source code:
  - Mailing list and Issue tracker
  - StackOverflow Discussions

[Panichella at al. - ICPC 2012]

[Vassallo at al. - ICPC 2014]

Software Changes over the Time…

*"…as consequence the original documentation tend to be **incomplete** and **inconsistent** with the source code…"*



API Document Defects are Frequent

"…and tend to be discovered and fixed after long time…"

stackoverflow

It does seem like the doc should say

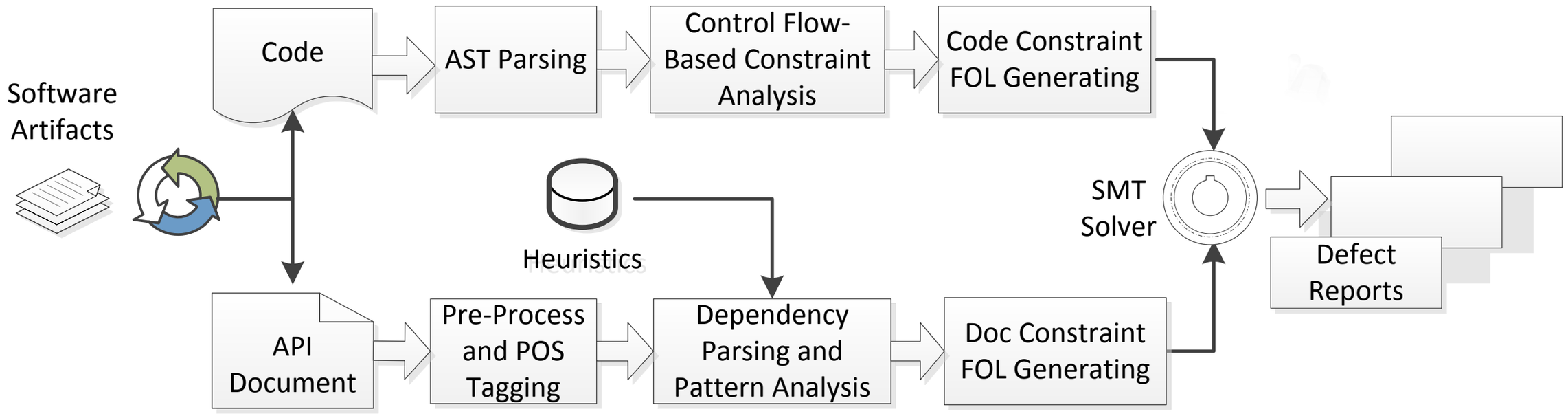
> After `cancel()` returns *true*, subsequent calls to `isDone()` ...

The Javadocs are often flat-out wrong and contradictory, and Sun has often not even bothered to fix them even after 10 years. Careful testing should always supplement the docs.

http://stackoverflow.com/questions/2967303/inconsistency-in-java-util-concurrent-future

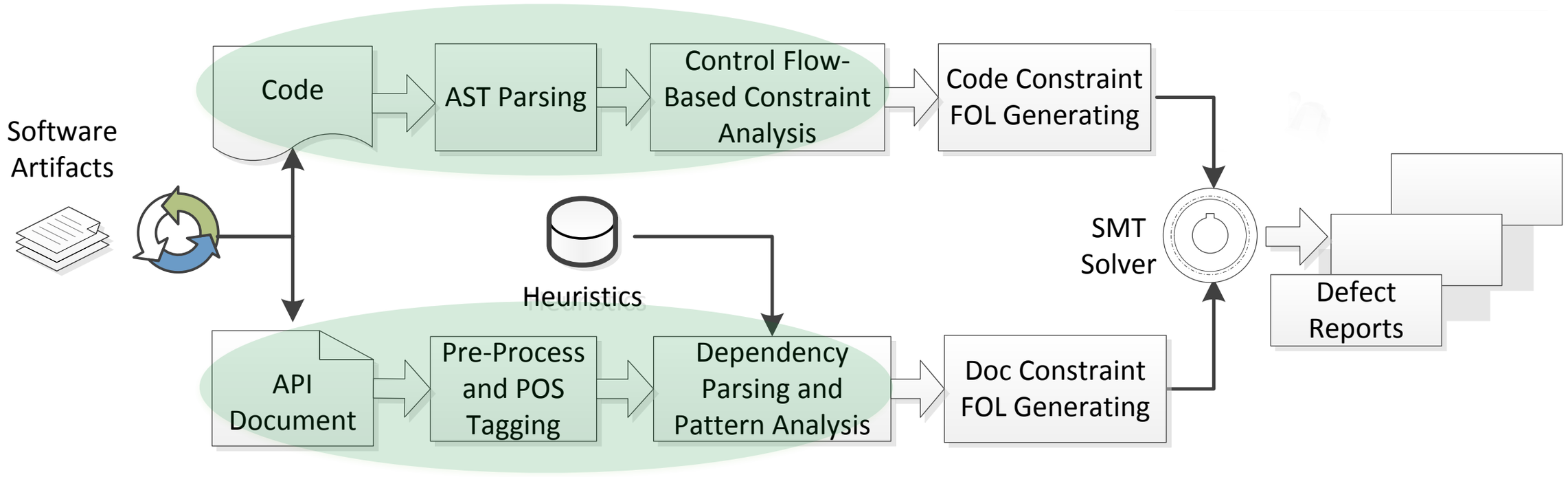
DRONE

DetectoR of dOcumentatioN dEfects



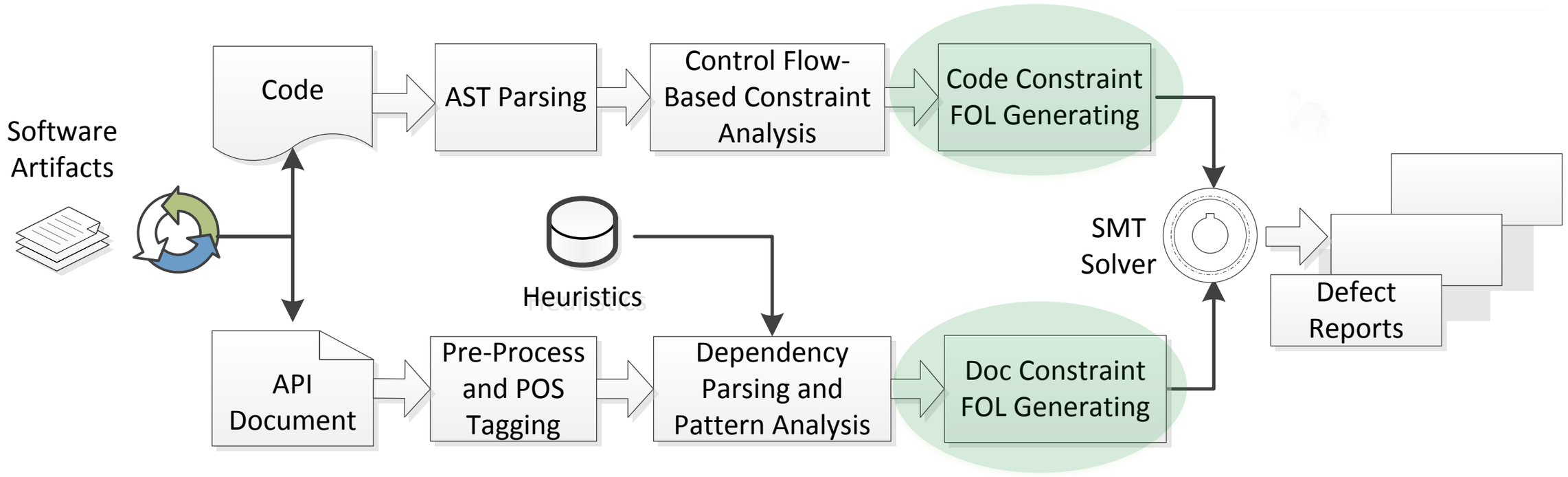
DRONE

DetectoR of dOcumentatioN dEfects



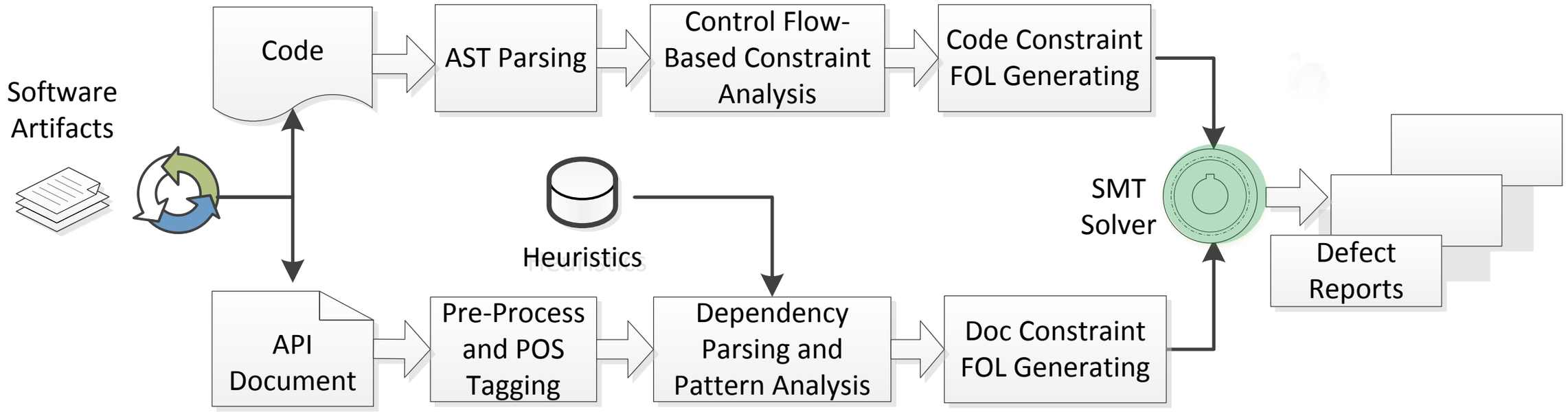
DRONE

DetectoR of dOcumentatioN dEfects

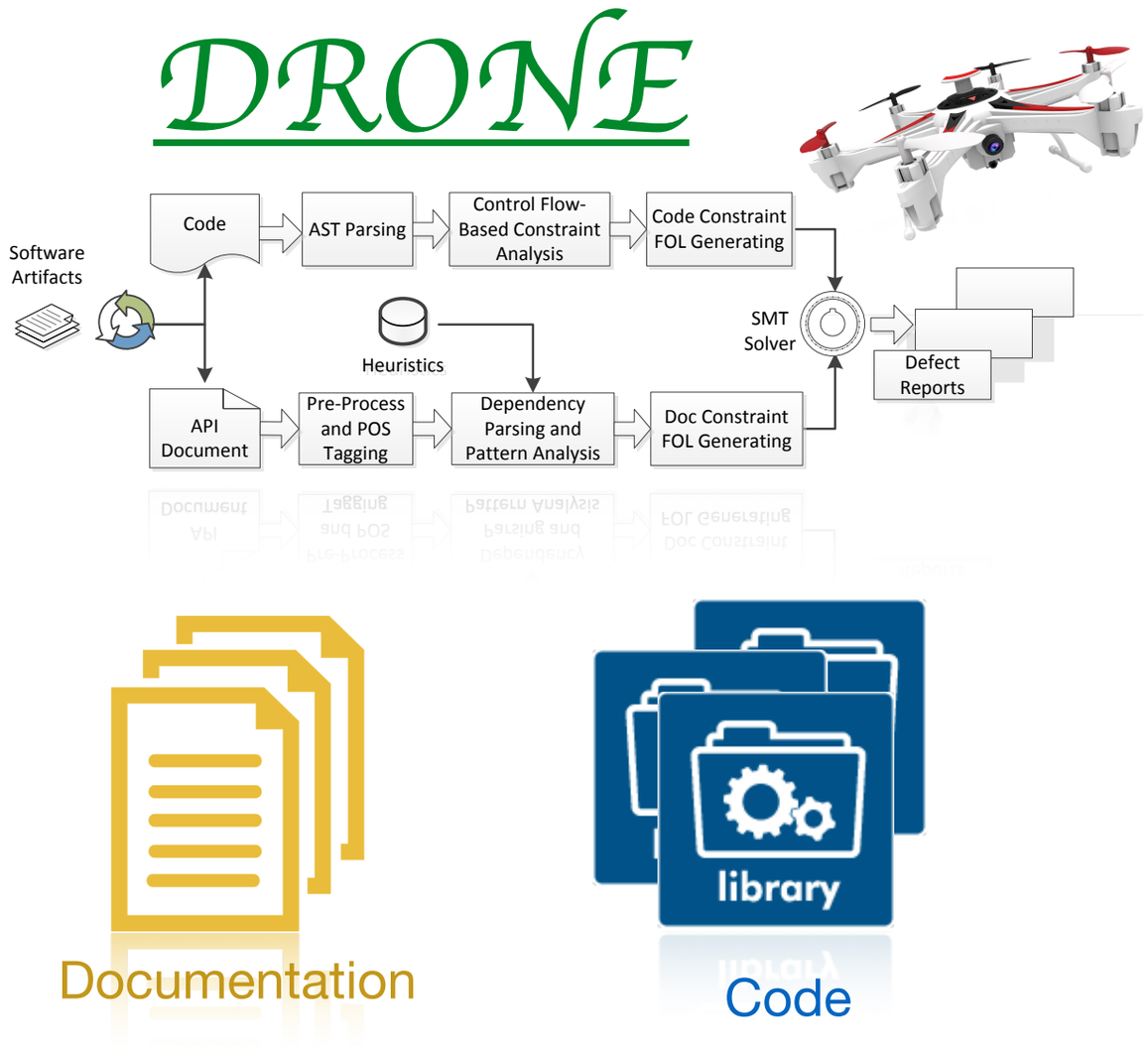


DRONE

DetectoR of dOcumentatioN dEfects



Evaluation of the Summaries



With *DRONE* we analyzed over **1 million** of **LOC** and more than **30,000 Javadoc documents** belonging to 8 java libraries

detecting around **2000** of **API documentation defects**.

with **high precision** (values between 0.58 - 0.83) **and** an high recall (values > 0.81) **results**.

"*Analyzing APIs Documentation and Code to Detect Directive Defects*". **ICSE 2017**

Release Cycle

Performance Testing	**Coding activities**	**A/B testing**
Microservices	**Maintenance activities**	**Structured e-mails**
CD/CI	**Code Review**	**Survey**
Automated testing	**Testing**	**App reviews**
Security	**Team coordination**	**Crowdsourcing**
…	**Software documentation**	**…**
	…	

Release Cycle

Performance Testing

Microservices

CD/CI

Automated testing

Coding activities

Maintenance activities

Code Review

Testing

A/B testing

Structured e-mails

Survey

App reviews

Security

...

Team coordination

Software documentation

...

Crowdsourcing

...

Release Cycle

Performance Testing

Microservices

CD/CI

Automated testing

Coding activities

Maintenance activities

Code Review

Testing

A/B testing

Structured e-mails

Survey

App reviews

Security

...

Team coordination

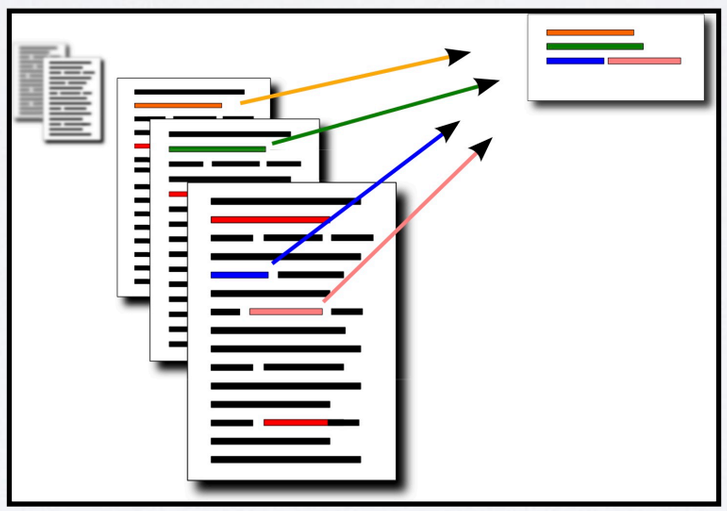
Software documentation

...

Crowdsourcing

...

Test Cases Summarization



Example of Test Case Generated by Evosuite

```
1| public class TestOption {
2|
3| @Test
4| public void test0() throws Throwable {
5|    Option option0 = new Option("", "1W|^");
6|    assertEquals("1W|^", option0.getDescription());
7|    assertEquals("", option0.getKey());
8|  }
9| }
```

Test Case Automatically Generated by Evosuite (for the class apache.commons.Option.Java)

Example of Test Case Generated by Evosuite

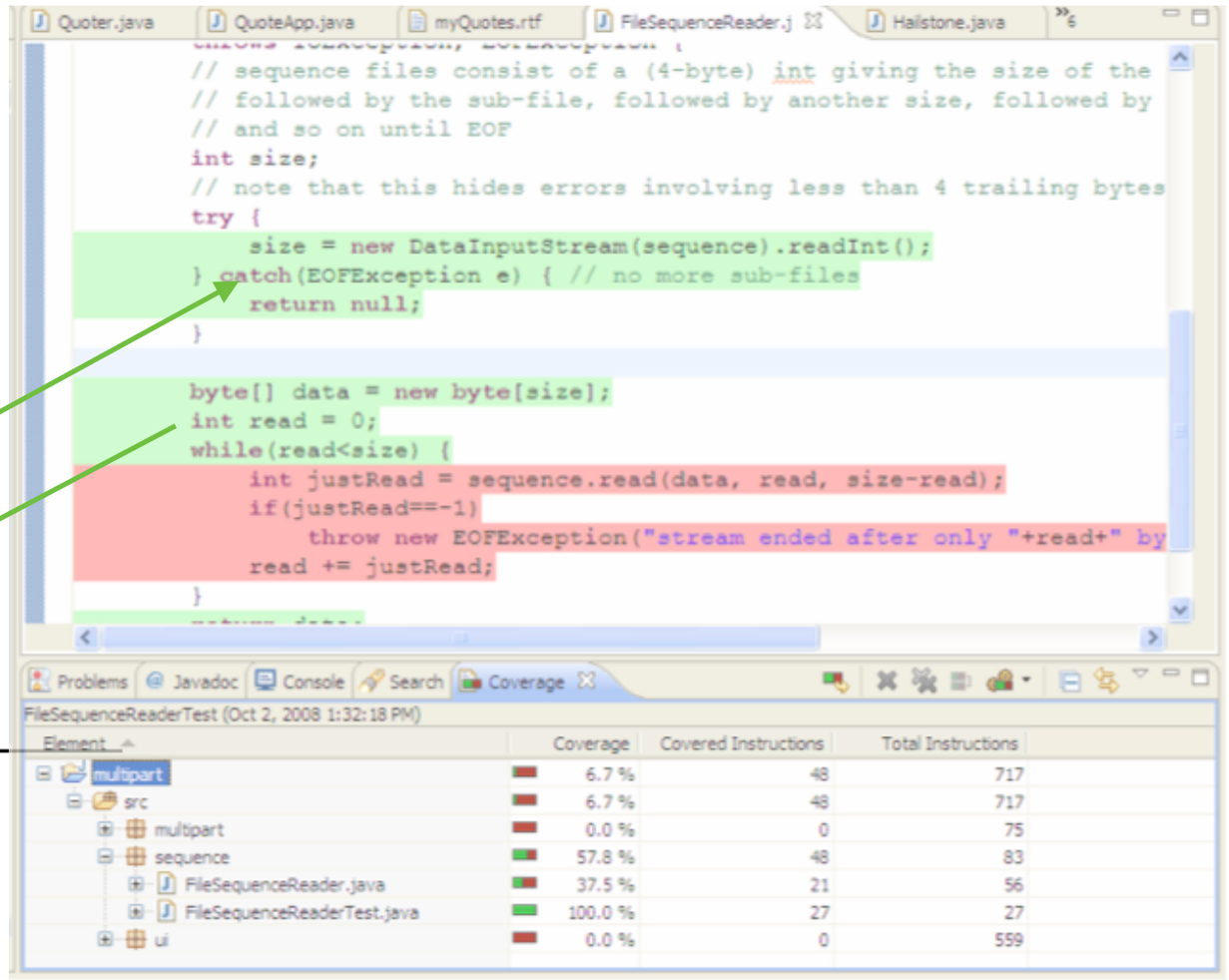
Not Meaningful
Names for Test Methods

```
1| public class TestOption {
2|
3| @Test
4| public void test0() throws Throwable {
5|    Option option0 = new Option("", "1W|^");
6|    assertEquals("1W|^", option0.getDescription());
7|    assertEquals("", option0.getKey());
8|  }
9| }
```

Test Case Automatically Generated by Evosuite (for the class apache.commons.Option.Java)

It is difficult to tell, without reading the contents of the target class, what is the behavior under test.

Example of Test Case Generated by Evosuite

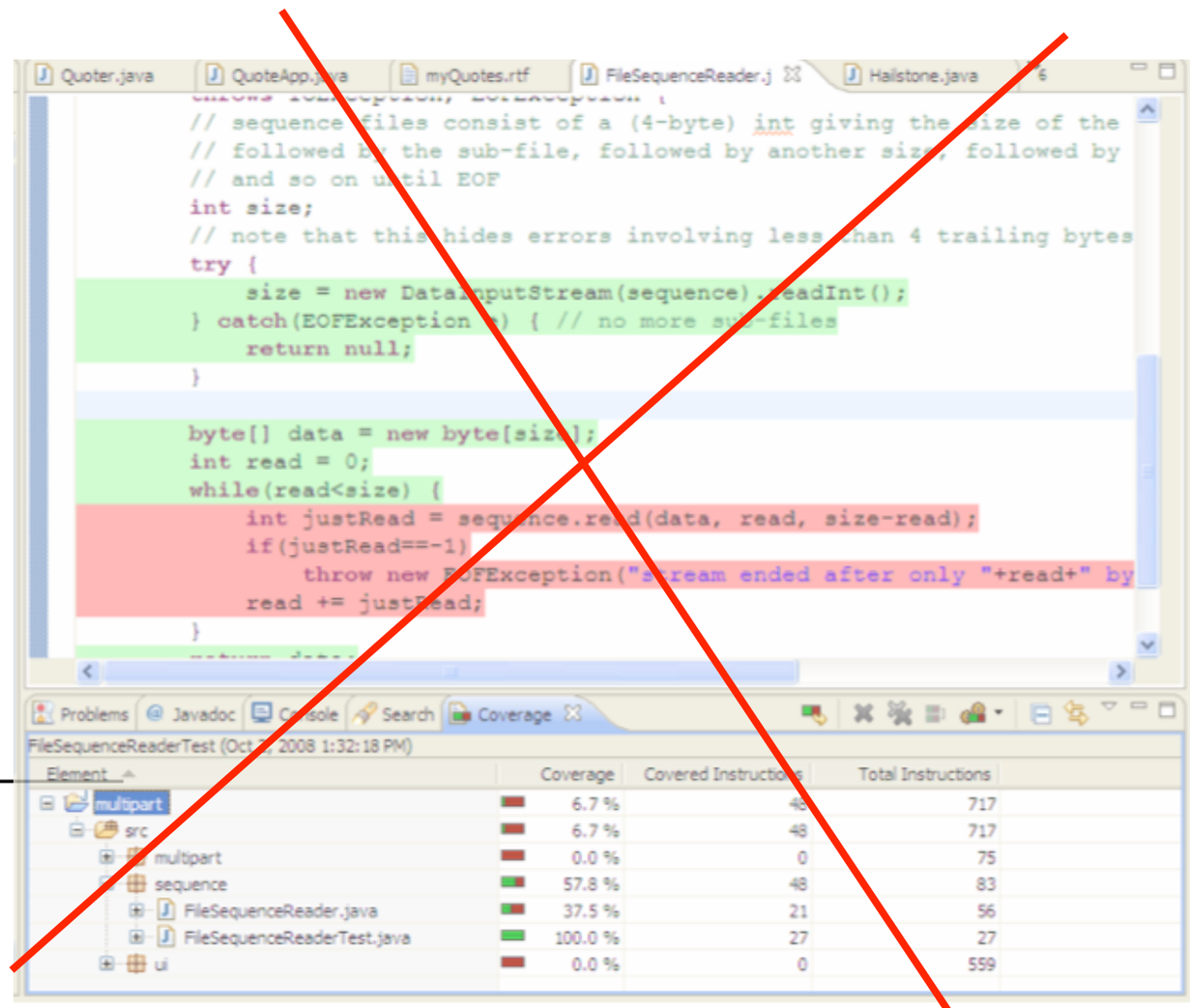



```
1| public class TestOption {
2|
3| @Test
4| public void test0() throws Throwable {
5|    Option option0 = new Option("", "1W|^");
6|    assertEquals("1W|^", option0.getDescription());
7|    assertEquals("", option0.getKey());
8|  }
9| }
```

Test Case Automatically Generated by Evosuite (for the class apache.commons.Option.Java)

Our Solution: Automatically Generate Summaries of Test Cases

```
 1| /** The main class under test is Option. It describes
 2|  * a single option and maintains information regarding:
 3|  *  - the option;
 4|  *  - the long option;
 5|  *  - the argument name;
 6|  *  - the description of the option;  3.a
 7|  *  - whether it has required;
 8|  *  - whether it has optional argument;
 9|  *  - the number of arguments;
10|  *  - the type, the values and the separator of the option;**/
11| public class TestOption {
12| /** OVERVIEW: The test case "test0" covers around 3.0% 3.b
13|  * (low percentage) of statements in "Option" **/
14| @Test
15| public void test0() throws Throwable {
16| // The test case instantiates an "Option" with option 3.c
17| // equal to "", and description equal to "1W|^".
18| Option option0 = new Option("", "1W|^");
19| // Then, it tests:
20| // 1)  whether the description of option0 is equal to 3.c
21| // "1W|^";
22|  assertEquals("1W|^", option0.getDescription());
23| // 2) whether the key of option0 is equal to ""; 3.c
24| // The execution of the method call used in the assertion 3.d
25| // implicitly covers the following 1 conditions:
26| // - the condition "option equal to null" is FALSE;
27|  assertEquals("", option0.getKey());
28|  }
29| }
```




```
1| public class TestOption {
2|
3| @Test
4| public void test0() throws Throwable {
5|    Option option0 = new Option("", "1W|^");
6|    assertEquals("1W|^", option0.getDescription());
7|    assertEquals("", option0.getKey());
8|  }
9| }
```

Test Case Automatically Generated by Evosuite (for the class apache.commons.Option.Java)

Results

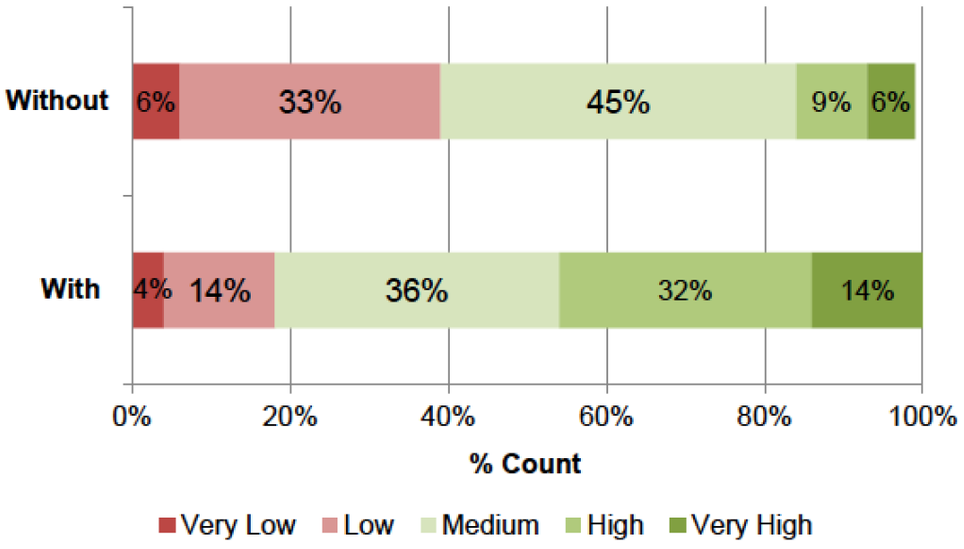
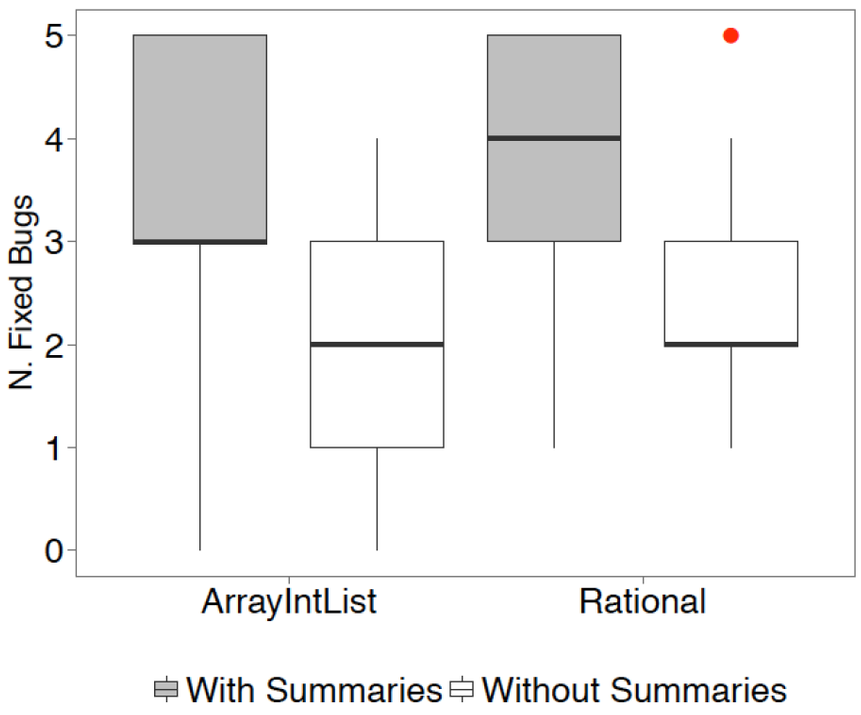


Figure 5: Perceived test comprehensibility WITH and WITHOUT TestDescriber summaries.



30 Developers:

- 22 Researchers

- 8 Professional Developers

Release Cycle

Performance Testing

Microservices

CD/CI

Automated testing

Coding Activities

Maintenance Activities

Code Review

Testing

A/B testing

Structured e-mails

Survey

App reviews

Security

...

...

Team coordination

Software documentation

...

Crowdsourcing

...

...

Release Cycle

Performance Testing

Microservices

CD/CI

Automated testing

Coding activities

Maintenance activities

Code Review

Testing

A/B testing

Structured e-mails

Survey

App reviews

Security

...

...

Team coordination

Software documentation

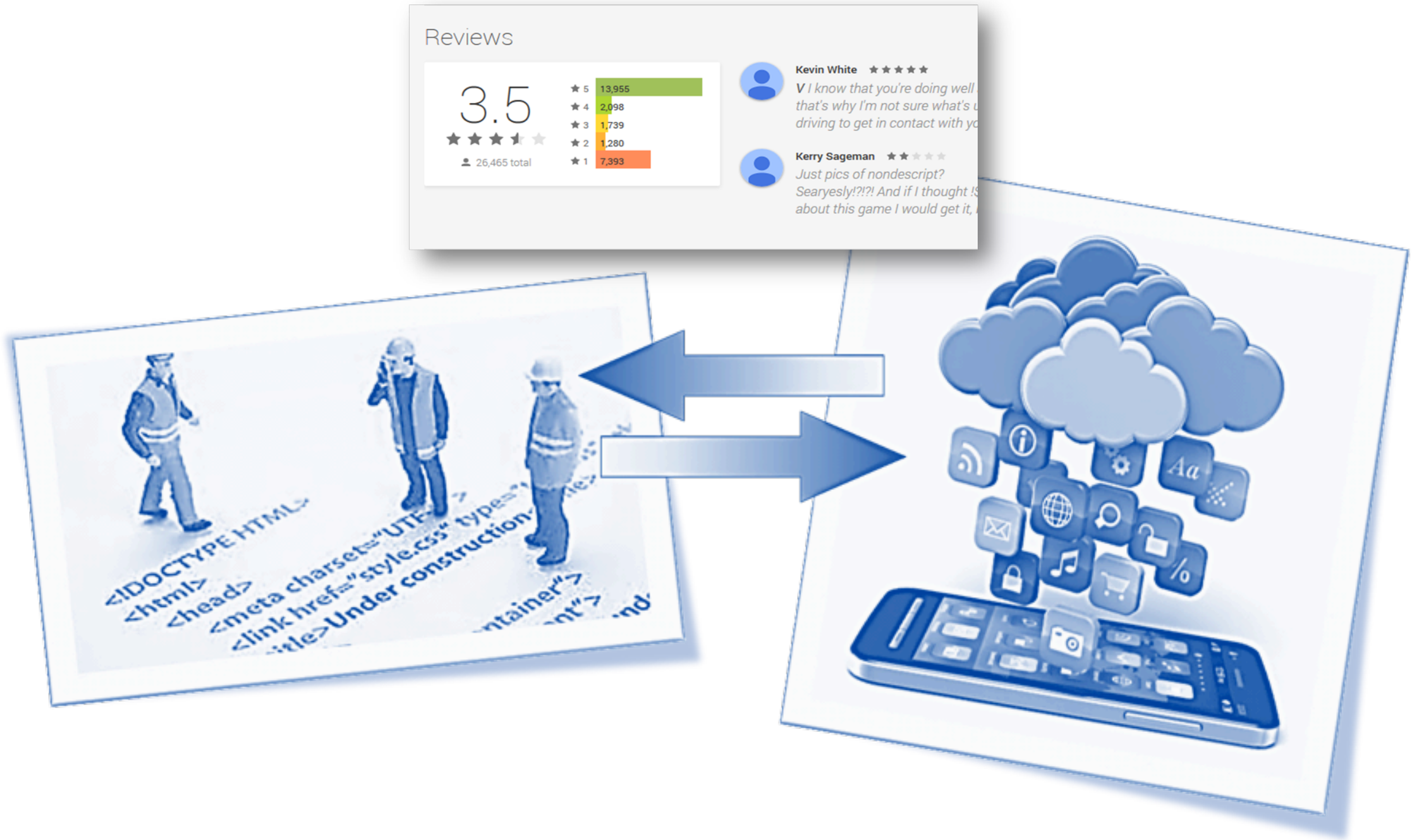
...

Crowdsourcing

...

...

Maintenance of Mobile Applications

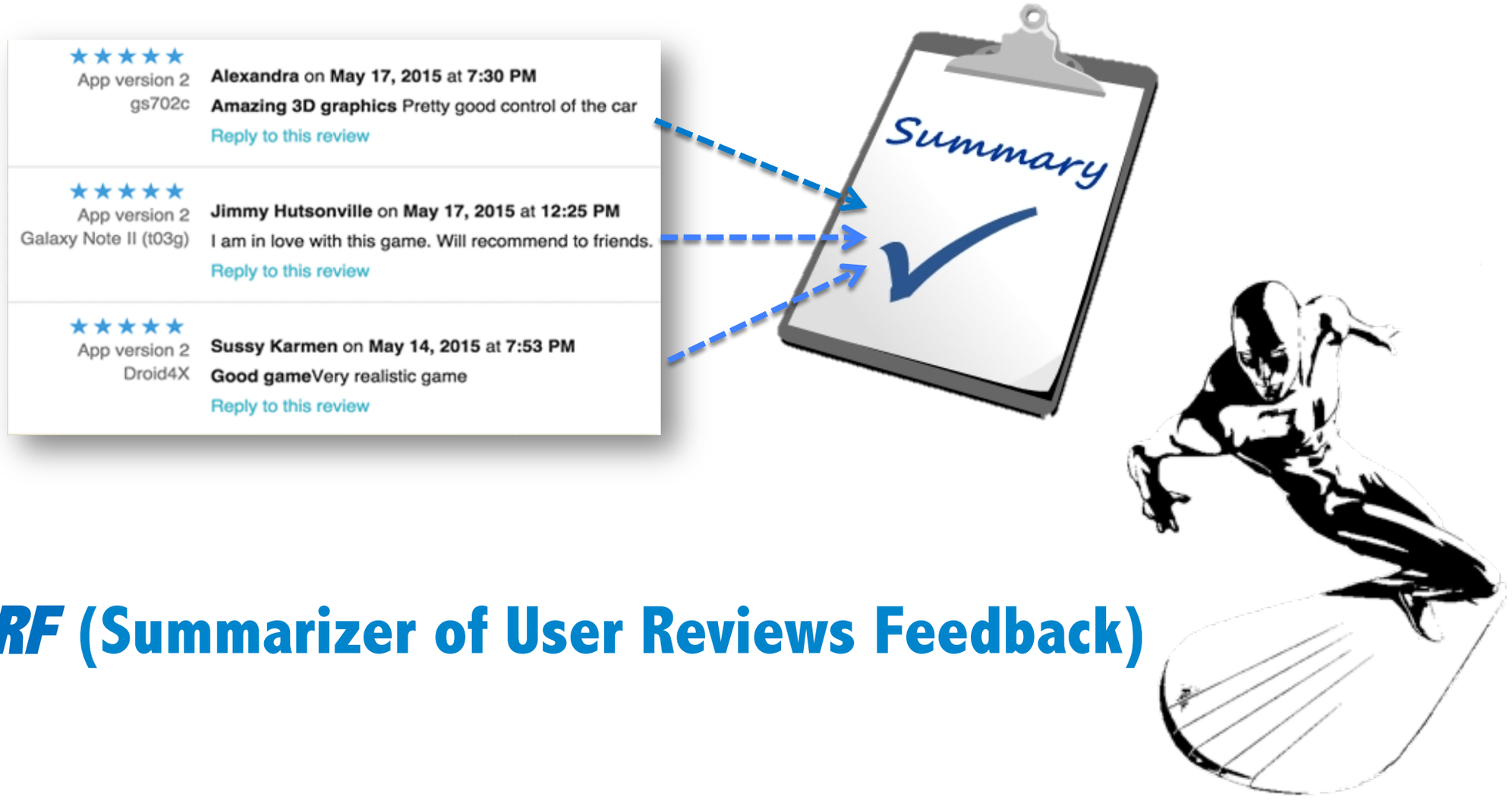


"*About one third of app reviews contain useful information for developers*"

Pagano et. al. RE2013

The Problem

Summaries of User Reviews



SURF (Summarizer of User Reviews Feedback)

What Would Users Change in My App? Summarizing App Reviews for Recommending Software Changes.
24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE) - 2016.

SURF: Summarizer of User Reviews Feedback.
Proceedings of the 39th IEEE International Conference on Software Engineering (ICSE) - 2017

Summary Generation

Review Summary for karaoke_paid_summary.xml

Click on a topic to show the issues related to that topic. The issues are grouped by review intent, i.e. **bugs**, **requests**, **information**, **questions** or **miscellaneous**. Click on the bars next to each topic to filter the sentences by intent.

When browsing a topic, click on a sentence to see the full review it was taken from. The sentence will be highlighted for context.

GUI	3 2 3	
BUG	But while playing, editing and flicking between screens it vanished	
BUG	Still playing the track with a BLACK screen and no response	
BUG	It has failed to play some tracks but there is sure to be another version among my 40,000 tracks	
REQUEST	Wish there was a way to get bigger fonts on the midi also the ability to have different background pictures fade in and out or even play a windows movie maker file in the background	
REQUEST	It would be better if there is a screen mirror option or chrome cast	
INFO	I can now tick midi player app off my bucket list and move on to the next item	
INFO	Great as a Karaoke machine and great for midi backing tracks	
INFO	Love it top dog program Love it	
Feature/Functionality	3 8 9	
Contents	3	
Model	1 3	
App	1 2 8	
Security	1	
Improvement	1	
Update/Version	1	

http://www.ifi.uzh.ch/en/seal/people/panichella/tools/SURFTool.html

SURF: Summarizer of User Reviews Feedback.
Proceedings of the 39th IEEE International Conference on Software Engineering (ICSE) - 2017

Case Study

Involving 23 Developers

2622 Reviews

Of

12 Apps

Results

V.S.

SURF helps to prevent more than half of the time required for analyzing users feedback and planning software changes.

92% of manually extracted feedback appears also in the automatic generated summaries.

Summaries generated by ***SURF*** are reasonably correct, adequate, concise, and expressive.

Conclusion

Performance Testing	**Coding Activities** ✓	**A/B testing**
Microservices	**Maintenance Activities** ✓	**Structured e-mails**
CD/CI	**Code Review**	**Survey**
Automated testing	**Testing** ✓	**App reviews** ✓
Security	**Team coordination**	**Crowdsourcing**
…	**Software documentatio** ✓	**…**
…	**…**	**…**

Conclusion

Performance Testing

Coding Activities ✓

A/B testing

Microservices

Maintenance Activities ✓

Structured e-mails ←

CD/CI ←

Code Review ←

Survey ←

Automated testing ←

Testing ✓

App reviews ✓

Security

Team coordination ←

Crowdsourcing ←

…

Software documentation ✓

…

…

…

…

Conclusion

Performance Testing

Coding Activities

A/B testing

Microservices

Maintenance Activities

Structured e-mails

CD/CI

Code Review

Survey

Automated testing

Testing

App reviews

Security

Team coordination

Crowdsourcing

…

Software documentation

…

…

…

…

Exploring the Integration of User Feedback in Automated Testing of Android Applications

***Abstract*—The intense competition characterizing mobile application's marketplaces forces developers to create and maintain high-quality mobile apps in order to ensure their commercial success and acquire new users. This motivated the research community to propose solutions that automate the testing process of mobile apps. However, the main problem of current testing tools is that they generate redundant and random inputs that are insufficient to properly simulate the human behavior, thus leaving feature and crash bugs undetected until they are encountered by users. To cope with this problem, we conjecture that information available in user reviews—that previous work showed as effective for maintenance and evolution problems—can be successfully exploited to identify the main issues users experience while using mobile applications, e.g., GUI problems and crashes. In this paper we provide initial insights into this direction, investigating (i) what type of user feedback can be actually exploited for testing purposes, (ii) how complementary user feedback and automated testing tools are, when detecting crash bugs or errors and (iii) whether an automated system able to monitor crash-related information reported in user feedback is sufficiently accurate. Results of our study, involving 11,296 reviews of 8 mobile applications, show that user feedback can be exploited to provide contextual details about errors or exceptions detected by automated testing tools. Moreover, they also help detecting bugs that would remain uncovered when rely on testing tools only. Finally, the accuracy of the proposed automated monitoring system demonstrates the feasibility of our vision, i.e., integrate user feedback into testing process.**

***Index Terms*—Automated Software Testing, Mobile Applications, User Reviews Analysis**

I. Introduction

Mobile devices, such as smartphones and tablets, acquired more and more a central role in everyday life in recent years [33]. Consequently, we witnessed an unprecedented growth of the app industry, with around 149 billions of mobile apps downloaded by September 2016 [50] and 12 million of developers maintaining them [14]. The growing competition characterizing mobile application marketplaces, like Google Play and the Apple App Store, ensures that only high quality apps stay on the market and gain users. This forces developers to deliver high quality apps, maintaining them through adequate software testing activities [28], [33]. However, mobile applications differ from traditional software, being structured around Graphical User Interface (GUI) events and activities [1], [36]. Therefore, they expose different kinds of bugs and suffer of a higher defect density compared to traditional desktop and server applications [22]. To support developers in building high-quality applications, the research community has recently developed novel techniques and tools to automate such a testing process [22], [28], [31], [37], [45]. Techniques for mobile testing generate, according to different strategies, UI and system events, like the tap on a button or an incoming notification. Such events are then transmitted to the application under test (AUT) with the aim of detecting *unhandled runtime exceptions*. If an exception occurs, such techniques typically save two different information: (i) the correspondent *stack trace* and (ii) the sequence of events that led to the crash [37]. Unfortunately, most of these tools suffer of three important limitations. Firstly, the reports they generate (composed as mentioned, by a stack trace and a sequence of inputs) lack contextual information and are difficult to understand and analyze [10], [24]. Secondly, they are able to detect only bugs that actually cause unhandled exceptions, thus possibly missing those not raising any. Third and most important limitation, current tools *"are not suited for generating inputs that require human intelligence (*e.g.*, constructing valid passwords, playing and winning a game, etc.)"* [28]. Indeed, the generation of such random inputs often results in a redundant sequences of events that are not able to simulate the human behavior. For this reason, such tools (i) are often not able to achieve high code coverage [12], [38] and (ii) might fail to detect bugs and crashes that are encountered by users [11], [28].

In this context, recent work demonstrate that it is possible to leverage information available in app reviews to identify the main problems encountered by users while using an app [13], [40], [43], [48], [51]. We believe that such information can be successfully exploited to overcome some of the limitations of state-of-the-art tools for automated testing of mobile apps. Specifically, we argue that the integration of user feedback into the testing process can (i) *complement* the capabilities of automated testing tools, by identifying bugs that they cannot reveal or (ii) *facilitate* the diagnosis of bugs or crashes (since users might describe the actions that led to a crash).

To better explain the motivations behind our work, we provide below two concrete examples taken from the dataset gathered for this study (detailed in Section II). In the first example, we report the content of a user review, related to the `com.danvelazco.fbwrapper` app, revealing a bug missed by widely adopted automated testing tools such as Monkey [18] and Sapienz [31]:

> *"Love the idea of this app but anytime I leave the page the screen goes completely white and won't come back until force-stopped. Update: I thought the white screen was because my phone was so outdated but it still does it on my Nexus 6 ...".*

In this case, the user complains for a white screen displayed after leaving the previous page in the application. However,

Giovanni Grano

G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, H. Gall
SANER - 2018

Thanks!