Engenharia de Software II

Mineração de Software: Introdução

Prof. André Hora
DCC/UFMG
2019.1

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG

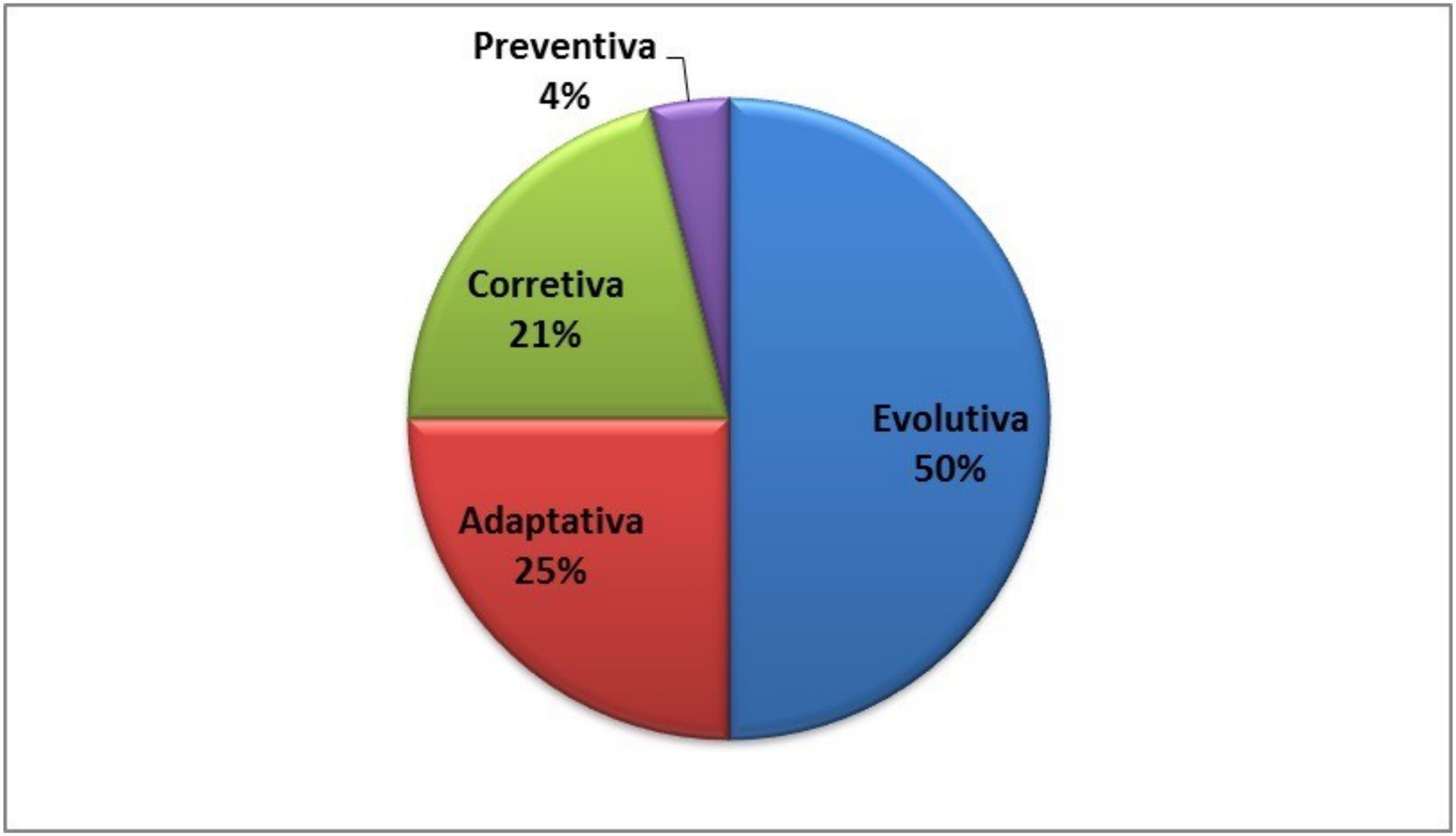
Evolução de Software

- Sistemas de software estão em constante evolução

- Novas funcionalidades, correção de bugs, refatoração de código

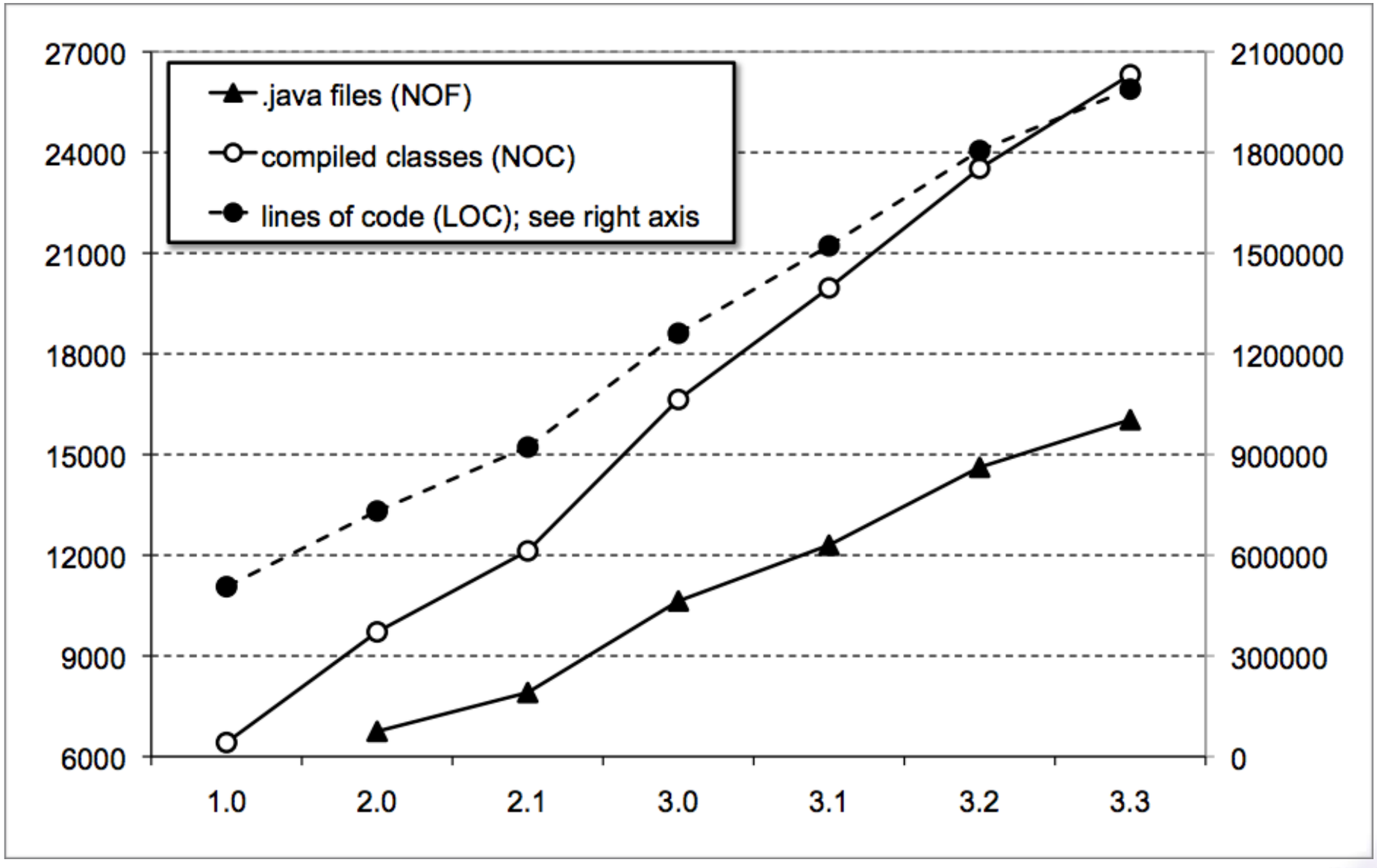
- Representa até **90%** dos custos de desenvolvimento

Proporção de Manutenção

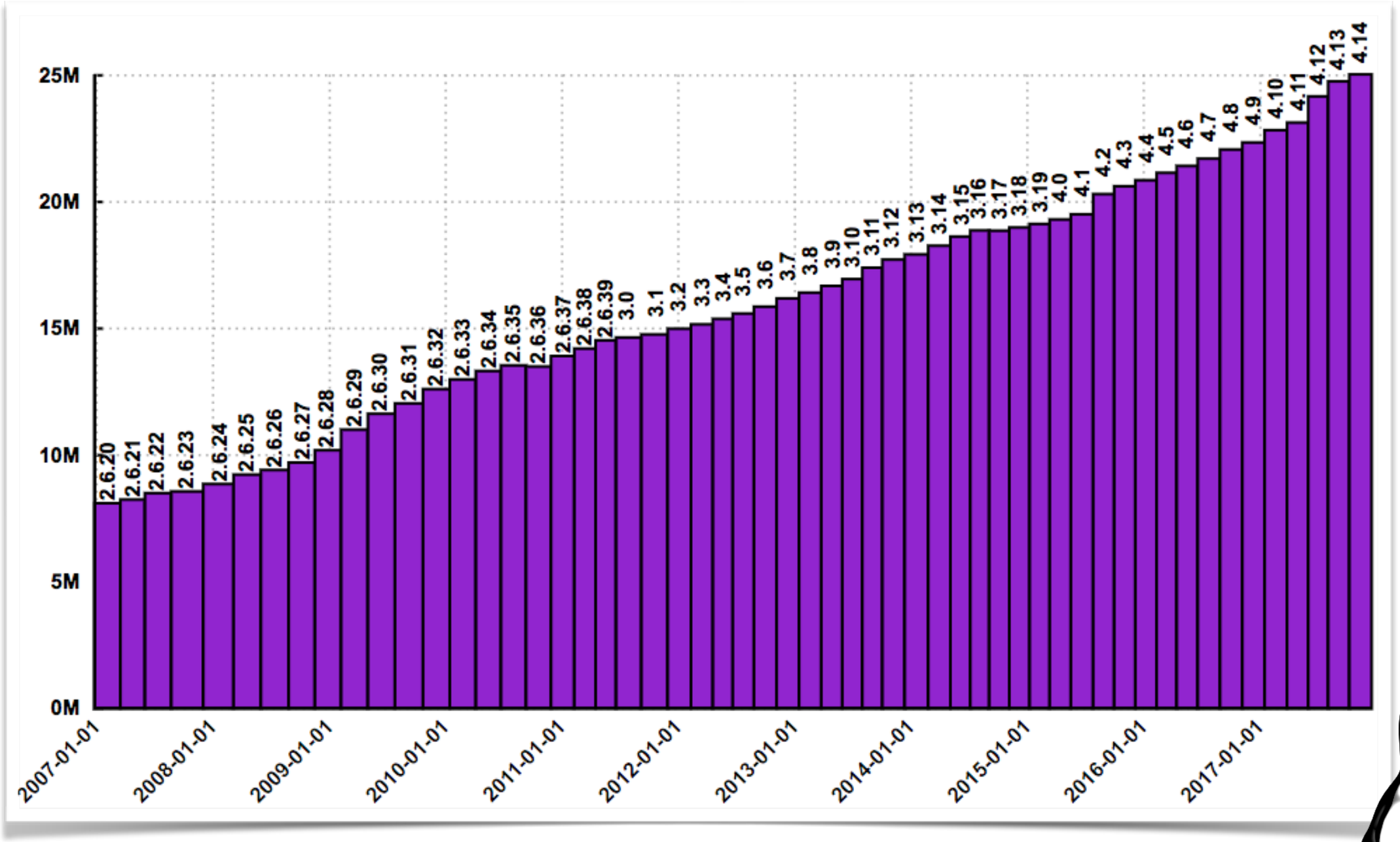


http://www.leandromtr.com/tecnologia-informacao/tipos-de-manutencao-de-software

Linhas de Código do **Eclipse**



Linhas de Código do **Linux**



Razões para Evolução de Software

Se um sistema é utilizando, ele nunca está finalizado pois precisa sempre evoluir para:

Acomodar novas funcionalidades

Corrigir defeitos

Melhorar design

Comunicar com outros sistemas

Migrar de SO, BD, bibliotecas, etc

Adaptar a diferentes hardware

Adaptar a leis, regras de negócio, etc

Refatorar código

Leis de Lehman

↑ Mudança contínua

↑ Complexidade crescente

↑ Crescimento contínuo

↓ Declínio da qualidade

Mineração de Software

Mineração de Software

- Repositórios de software contém uma fonte rica de informações do estado **atual** e do seu **passado**
- Utilizando técnicas voltadas para mineração de repositórios de software:
  - **Desenvolvedores:** adquirem maior conhecimento sobre como manter e evoluir sistemas de software
  - **Pesquisadores:** exploraram esses dados para entender e melhorar as práticas de desenvolvimento

Mineração de Software

- Ajuda a melhorar nossa compreensão sobre o desenvolvimento de software
- Pode contribuir para a criação de novos métodos e de uma nova geração de ferramentas na ES

- Exemplos:
  - Detecção de códigos candidatos a refatoração
  - Suporte à migração de bibliotecas de software
  - Sugestão de melhoria de modularidade
  - Suporte a melhoria de documentação, entre vários outros…

Engenharia de Software II

Table I.1. The 15 SWEBOK KAs
Software Requirements
Software Design
Software Construction
Software Testing
Software Maintenance
Software Configuration Management
Software Engineering Management
Software Engineering Process
Software Engineering Models and Methods
Software Quality
Software Engineering Professional Practice
Software Engineering Economics
Computing Foundations
Mathematical Foundations
Engineering Foundations

Mineração de Software

Quais artefatos podem ser analisados?

Quais artefatos podem ser analisados?

Dados (ex, código, documentação)

Quais artefatos podem ser analisados?

Dados (ex, código, documentação)

Metadados (ex, autor, avaliações de um app, downloads)

Quais artefatos podem ser analisados?

Dados (ex, código, documentação)

Histórico de Versões (ex, releases, commits)

Metadados (ex, autor, avaliações de um app, downloads)

Dados (eg, código, documentação)

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

```
/**
 * Simple HelloButton() method.
 * @version 1.0
 * @author john doe <doe.j@example.com>
 */
HelloButton()
{
  JButton hello = new JButton( "Hello, wor
  hello.addActionListener( new HelloBtnList

  // use the JFrame type until support for t
  // new component is finished
  JFrame frame = new JFrame( "Hello Button"
  Container pane = frame.getContentPane();
  pane.add( hello );
  frame.pack();
  frame.show();           // display the fra
}
```

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.baeldung.javadoc

Class SuperHero

java.lang.Object
  com.baeldung.javadoc.Person
    com.baeldung.javadoc.SuperHero

```
public class SuperHero
extends Person
```

Hero is the main entity we will be using to . . .

Author:

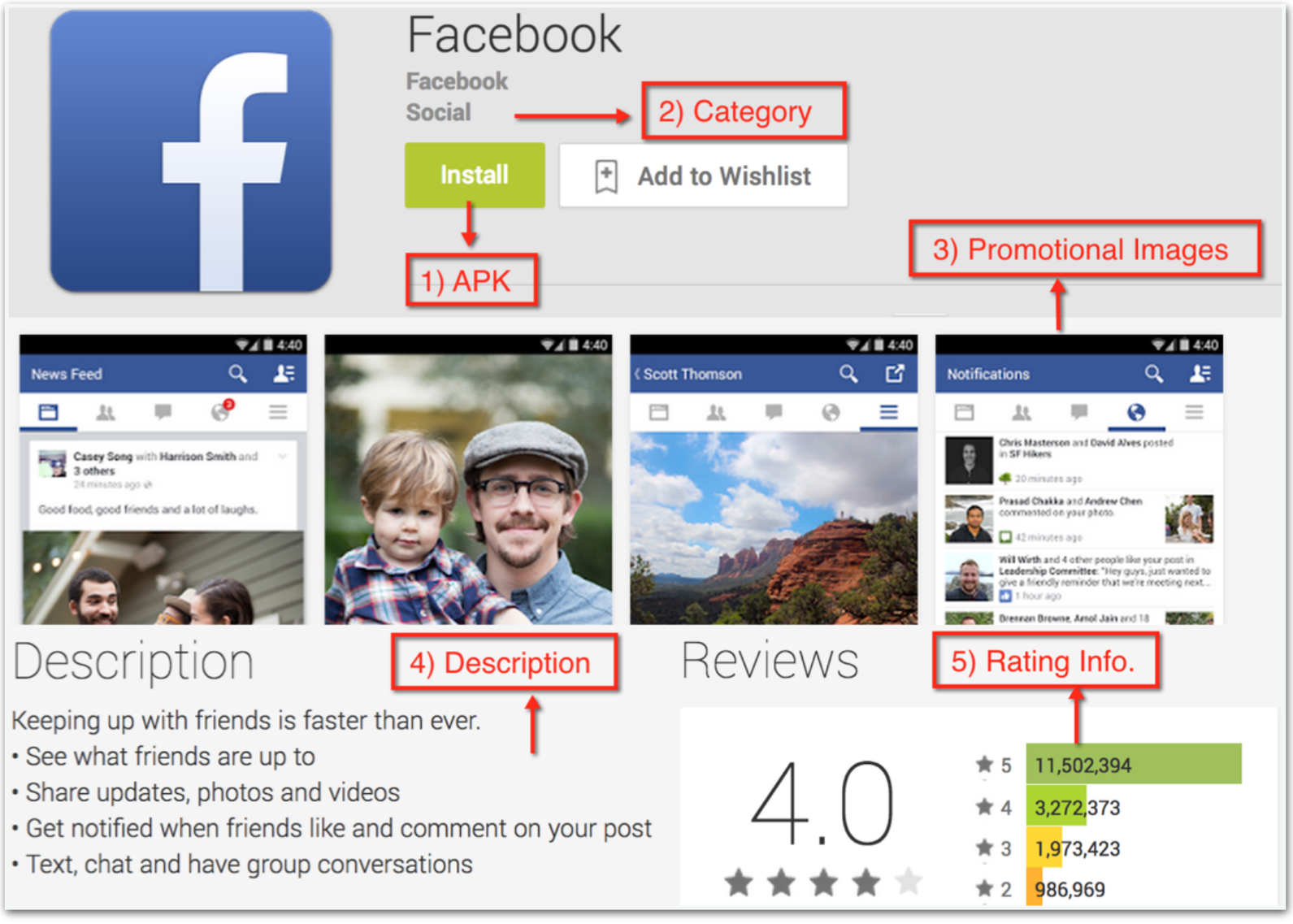
Captain America

Field Summary

Fields

Modifier and Type	Field and Description
private int	defense
private int	health
private java.lang.String	heroName The public name of a hero that is common knowledge
private java.lang.String	uniquePower

Metadados
(eg, autor, avaliações de um app, downloads)



Metadados (eg, autor, avaliações de um app, downloads)

Change sort order of shard stats in CCR test

Browse files

```
This commit changes the sort order of shard stats that are collected in
CCR retention lease integration tests. This change is done so that
primaries appear first in sort order.
```

master (#140)

jasontedor committed 2 days ago Verified

1 parent 91f69fc commit 14510a933a0b1ecd38033ecc21f2621eba45698a

Showing 1 changed file with 1 addition and 1 deletion.

Unified Split

2 x-pack/plugin/ccr/src/test/java/org/elasticsearch/xpack/ccr/CcrRetentionLeaseIT.java View file

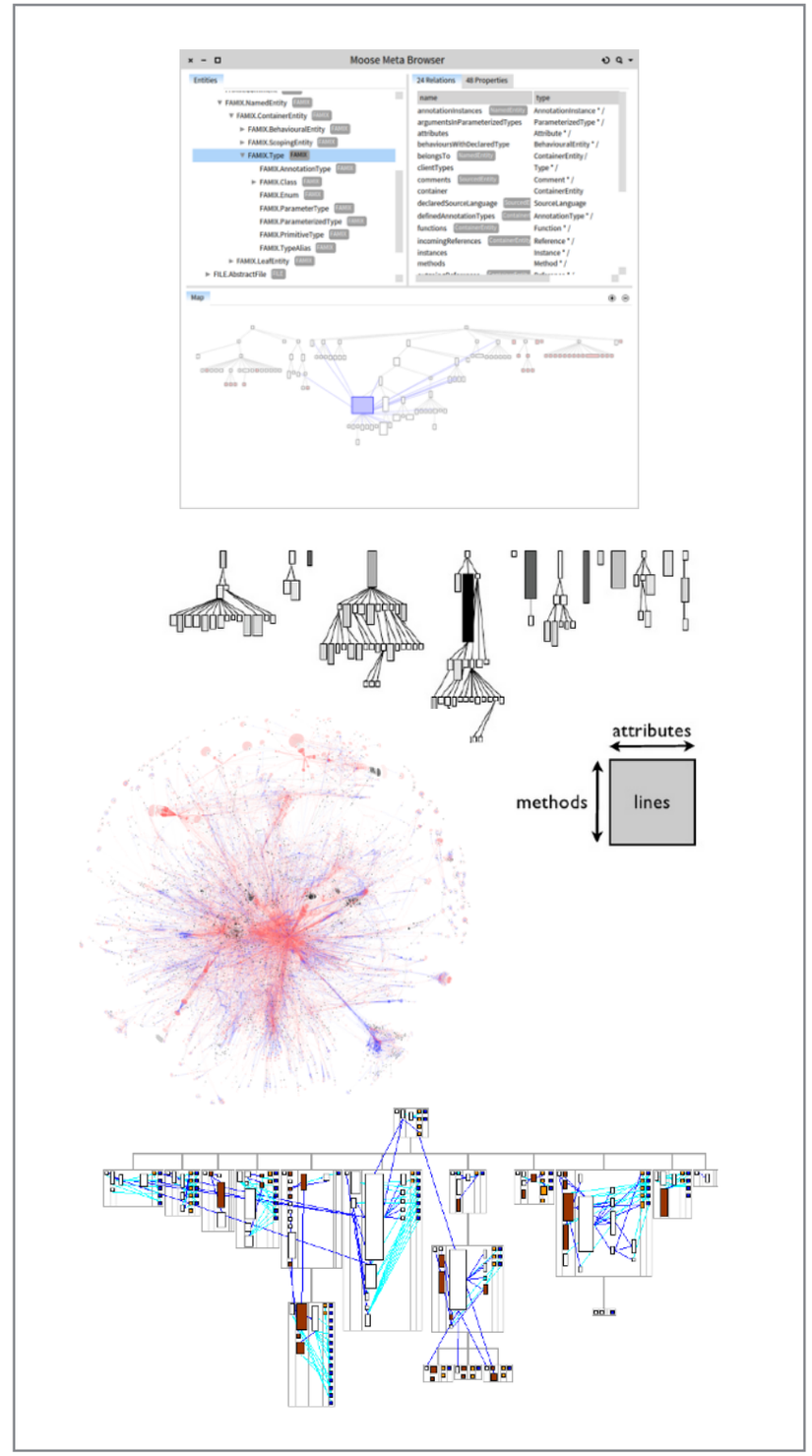
```
@@ -568,7 +568,7 @@ public void testUnfollowFailsToRemoveRetentionLeases() throws Exception {
568         return Arrays.stream(stats.getShards())
569                 .sorted((s, t) -> {
570                     if (s.getShardRouting().shardId().id() ==
    t.getShardRouting().shardId().id()) {
571 -                       return
    Boolean.compare(s.getShardRouting().primary(),
    t.getShardRouting().primary());
572                     } else {
573                         return
    Integer.compare(s.getShardRouting().shardId().id(),
    t.getShardRouting().shardId().id());
574                     }
```

```
568         return Arrays.stream(stats.getShards())
569 +               .sorted((s, t) -> {
570                     if (s.getShardRouting().shardId().id() ==
    t.getShardRouting().shardId().id()) {
571 +                       return -
    Boolean.compare(s.getShardRouting().primary(),
    t.getShardRouting().primary());
572                     } else {
573                         return
    Integer.compare(s.getShardRouting().shardId().id(),
    t.getShardRouting().shardId().id());
574                     }
```

Histórico de Versões (eg, releases, commits)


```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

Mineração de Software



Exemplos

Dê uma olhada

Ver todas as 2 imagens

Engenharia de Software (Português) Capa Comum – 15 jun 2011

por Ian Sommerville (Autor)

★★★☆☆ 8 avaliações de clientes

Ver todos os 3 formatos e edições

eBook Kindle	Capa Comum
R$ 149,10	**R$ 156,95**
Leia com nossos **apps gratuitos**	11 Novo(s) a partir de R$ 156,95

Em até 5x R$ 31,39 sem juros Calculadora de prestações

Entrega para o CEP 01319-900 na Sexta-feira, 1 de Setembro, se você finalizar o pedido em **23 horas e 20 minutos**

Leia Enquanto Enviamos

Leia Enquanto Enviamos

Compre e comece a ler a amostra digital deste livro enquanto espera ele chegar. Saiba mais aqui.

No intuito de atender às necessidades de alunos e professores dos cursos de ciência da computação, engenharia de computação e sistema de informação, esta nona edição de Engenharia de software teve seu conteúdo reestruturado e totalmente atualizado. A obra conta agora com novos capítulos que foca lizam o desenvolvimento ágil de software e os sistemas embarcados, além de trazer novas abordagens sobre engenharia dirigida a modelos, desenvolvimento open source, modelo Swiss Cheese de Reason,

Leia mais

Frequentemente comprados juntos

Preço total: **R$ 308,93**

Adicionar ambos ao carrinho

- ☑ **Este item:** Engenharia de Software por Ian Sommerville Capa comum R$ 156,95
- ☑ Engenharia de Software. Uma Abordagem Profissional por Roger S. Pressman Capa comum R$ 151,98

Dê uma olhada

Ver todas as 2 imagens

Engenharia de Software (Português) Capa Comum – 15 jun 2011

por Ian Sommerville (Autor)

★★★☆☆ 8 avaliações de clientes

Ver todos os 3 formatos e edições

eBook Kindle	**Capa Comum**
R$ 149,10	**R$ 156,95**
Leia com nossos **apps gratuitos**	11 Novo(s) a partir de R$ 156,95

Em até 5x R$ 31,39 sem juros Calculadora de prestações

Entrega para o CEP 01319-900 na Sexta-feira, 1 de Setembro, se você finalizar o pedido em **23 horas e 20 minutos**

Leia Enquanto Enviamos

Leia Enquanto Enviamos

Compre e comece a ler a amostra digital deste livro enquanto espera ele chegar. Saiba mais aqui.

No intuito de atender às necessidades de alunos e professores dos cursos de ciência da computação, engenharia de computação e sistema de informação, esta nona edição de Engenharia de software teve seu conteúdo reestruturado e totalmente atualizado. A obra conta agora com novos capítulos que foca lizam o desenvolvimento ágil de software e os sistemas embarcados, além de trazer novas abordagens sobre engenharia dirigida a modelos, desenvolvimento open source, modelo Swiss Cheese de Reason,

Leia mais

Frequentemente comprados juntos

Preço total: **R$ 308,93**

Adicionar ambos ao carrinho

- **Este item:** Engenharia de Software por Ian Sommerville Capa comum **R$ 156,95**
- Engenharia de Software. Uma Abordagem Profissional por Roger S. Pressman Capa comum **R$ 151,98**

Data mining

Dê uma olhada

Ver todas as 2 imagens

Engenharia de Software (Português) Capa Comum – 15 jun 2011

por Ian Sommerville (Autor)

★★★☆☆ 8 avaliações de clientes

Ver todos os 3 formatos e edições

eBook Kindle	**Capa Comum**
R$ 149,10	**R$ 156,95**
Leia com nossos **apps gratuitos**	11 Novo(s) a partir de R$ 156,95

Em até 5x R$ 31,39 sem juros Calculadora de prestações

Entrega para o CEP 01319-900 na Sexta-feira, 1 de Setembro, se você finalizar o pedido em **23 horas e 20 minutos**

Leia Enquanto Enviamos

Leia Enquanto Enviamos

Compre e comece a ler a amostra digital deste livro enquanto espera ele chegar. Saiba mais aqui.

No intuito de atender às necessidades de alunos e professores dos cursos de ciência da computação, engenharia de computação e sistema de informação, esta nona edição de Engenharia de software teve seu conteúdo reestruturado e totalmente atualizado. A obra conta agora com novos capítulos que foca lizam o desenvolvimento ágil de software e os sistemas embarcados, além de trazer novas abordagens sobre engenharia dirigida a modelos, desenvolvimento open source, modelo Swiss Cheese de Reason,

Leia mais

Frequentemente comprados juntos

Preço total: **R$ 308,93**

Adicionar ambos ao carrinho

- **Este item:** Engenharia de Software por Ian Sommerville Capa comum R$ 156,95
- Engenharia de Software. Uma Abordagem Profissional por Roger S. Pressman Capa comum R$ 151,98

Data mining

Sommerville → Pressman

Podemos utilizar essa técnica no contexto de software?

Regras de uso de APIs

```
234                                 String lock = unlock.getText().toString();
235                                 lock = lock.substring(0, lock.length() - 1);
236                                 unlock.setText(lock);
...
327             mHomeKeyLocker.lock(this);
328
329             }
330
331
332             unlock.setFilters(new InputFilter[] { new InputFilter.LengthFilter(12)
        });
```

before ➜ after

lock() ➜ unlock()

openFile() ➜ closeFile()

openStream() ➜ closeStream()

Regras de uso de APIs

```
234                                 String lock = unlock.getText().toString();
235                                 lock = lock.substring(0, lock.length() - 1);
236                                 unlock.setText(lock);
...
327             mHomeKeyLocker.lock(this);
328
329             }
330
331
332             unlock.setFilters(new InputFilter[] { new InputFilter.LengthFilter(12)
        });
```

before ➜ after

lock() ➜ unlock()

openFile() ➜ closeFile()

openStream() ➜ closeStream()

Qual a aplicação real?

Regras de uso de APIs

```
234                                 String lock = unlock.getText().toString();
235                                 lock = lock.substring(0, lock.length() - 1);
236                                 unlock.setText(lock);
...
327             mHomeKeyLocker.lock(this);
328
329             }
330
331
332             unlock.setFilters(new InputFilter[] { new InputFilter.LengthFilter(12)
        });
```

before ➜ after

lock() ➜ unlock()

openFile() ➜ closeFile()

openStream() ➜ closeStream()

Qual a aplicação real?

- **Melhorar documentação**
- **Detectar bugs**

Regras de evolução de APIs

```
 package org.jboss.as.modcluster;

-import junit.framework.Assert;
 import org.jboss.as.controller.PathAddress;
 import org.jboss.dmr.ModelNode;
 import org.jboss.dmr.ModelType;
+import org.junit.Assert;
 import org.junit.Test;
```

```
   public Collection<FormulaData> getChildren() {
-    List<FormulaData> result = new ArrayList<FormulaData>();
+    List<FormulaData> result = Lists.newArrayList();
     for (DecoratorContext childContext : decoratorContext.getChildren()) {
       result.add(new DefaultFormulaData(childContext));
     }
```

old ➜ new

ArrayList() ➜ List.newArrayList()

Vector ➜ ArrayList

junit.framework.Assert ➜ org.junit.Assert

Regras de evolução de APIs

```
 package org.jboss.as.modcluster;

-import junit.framework.Assert;
 import org.jboss.as.controller.PathAddress;
 import org.jboss.dmr.ModelNode;
 import org.jboss.dmr.ModelType;
+import org.junit.Assert;
 import org.junit.Test;
```

```
   public Collection<FormulaData> getChildren() {
-    List<FormulaData> result = new ArrayList<FormulaData>();
+    List<FormulaData> result = Lists.newArrayList();
     for (DecoratorContext childContext : decoratorContext.getChildren()) {
       result.add(new DefaultFormulaData(childContext));
     }
```

Qual a aplicação real?

old ➜ new

ArrayList() ➜ List.newArrayList()

Vector ➜ ArrayList

junit.framework.Assert ➜ org.junit.Assert

Regras de evolução de APIs

```
 package org.jboss.as.modcluster;

-import junit.framework.Assert;
 import org.jboss.as.controller.PathAddress;
 import org.jboss.dmr.ModelNode;
 import org.jboss.dmr.ModelType;
+import org.junit.Assert;
 import org.junit.Test;
```

```
   public Collection<FormulaData> getChildren() {
-    List<FormulaData> result = new ArrayList<FormulaData>();
+    List<FormulaData> result = Lists.newArrayList();
     for (DecoratorContext childContext : decoratorContext.getChildren()) {
       result.add(new DefaultFormulaData(childContext));
     }
```

old ➜ new

ArrayList() ➜ List.newArrayList()

Vector ➜ ArrayList

junit.framework.Assert ➜ org.junit.Assert

Qual a aplicação real?

- **Melhorar documentação**
- **Migrar bibliotecas**

Mais exemplos?

Pesquisas em MSR

Mineração de Software

- Repositórios de software contém uma fonte rica de informações do estado **atual** e do seu **passado**
- Utilizando técnicas voltadas para mineração de repositórios de software:
  - **Desenvolvedores:** adquirem maior conhecimento sobre como manter e evoluir sistemas de software
  - **Pesquisadores:** exploraram esses dados para entender e melhorar as práticas de desenvolvimento

Conferências

- **MSR**: International Conference on Mining Software Repositories
- **ICSE**: ACM/IEEE International Conference on Software Engineering
- **ICSME**: IEEE International Conference on Software Maintenance and Evolution
- **SANER**: IEEE International Conference on Software Analysis, Evolution and Reengineering
- **VEM**: Workshop on Software Visualization, Evolution and Maintenance

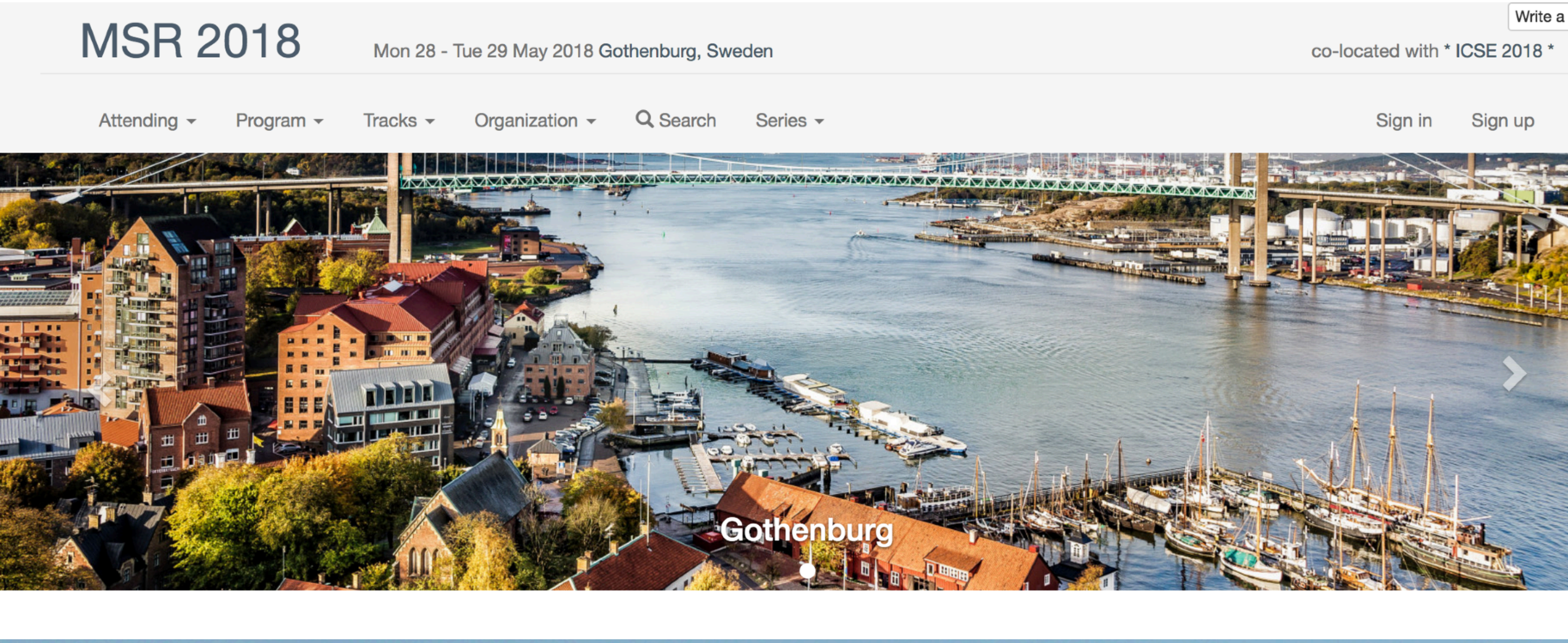
Write a

MSR 2018

Mon 28 - Tue 29 May 2018 Gothenburg, Sweden

co-located with * ICSE 2018 *

Attending Program Tracks Organization Search Series Sign in Sign up


MSR 2018
Mon 28 - Tue 29 May 2018 Gothenburg, Sweden
co-located with * ICSE 2018 *
Write a
Attending
Program
Tracks
Organization
Search
Series
Sign in
Sign up
Gothenburg


35th IEEE International Conference on
Software Maintenance and Evolution
Cleveland, OH USA
September 30th - October 4th 2019

MSR 2018 Mon 28 - Tue 29 May 2018 Gothenburg, Sweden co-located with * ICSE 2018 *

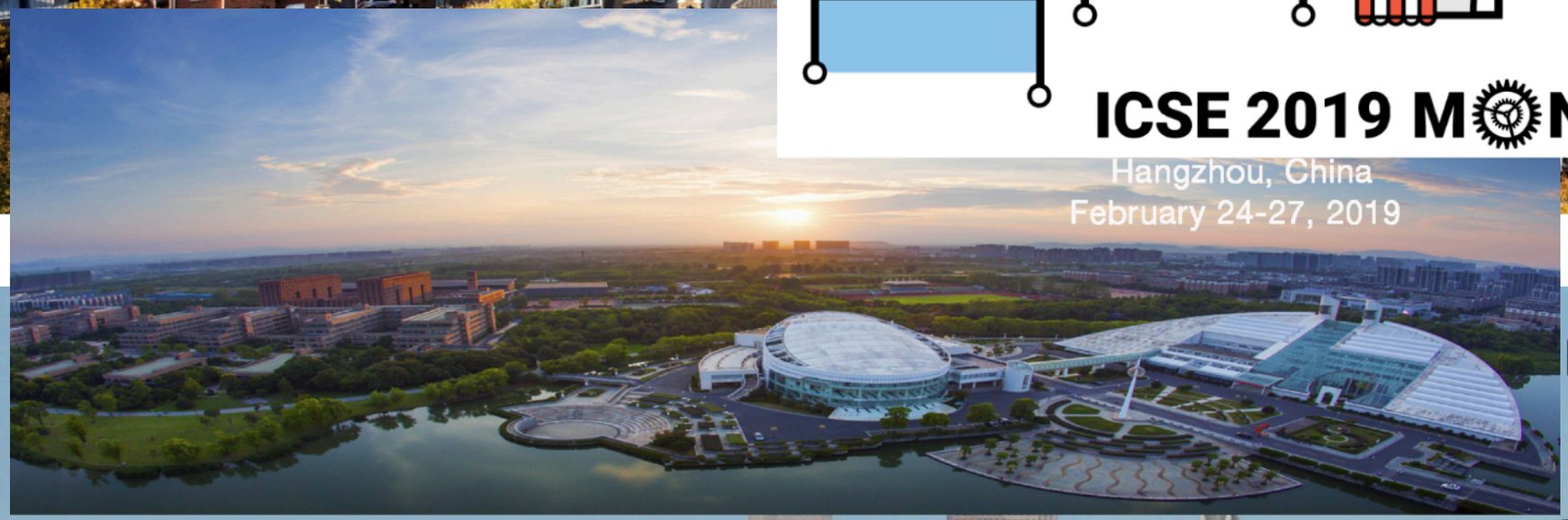
Write a

Attending Program Tracks Organization Search Series Sign in Sign up

MSR 2018 Mon 28 - Tue 29 May 2018 Gothenburg, Sweden co-located with * ICSE 2018 *

Attending Program Tracks Organization Search Series Sign in Sign up

Write a



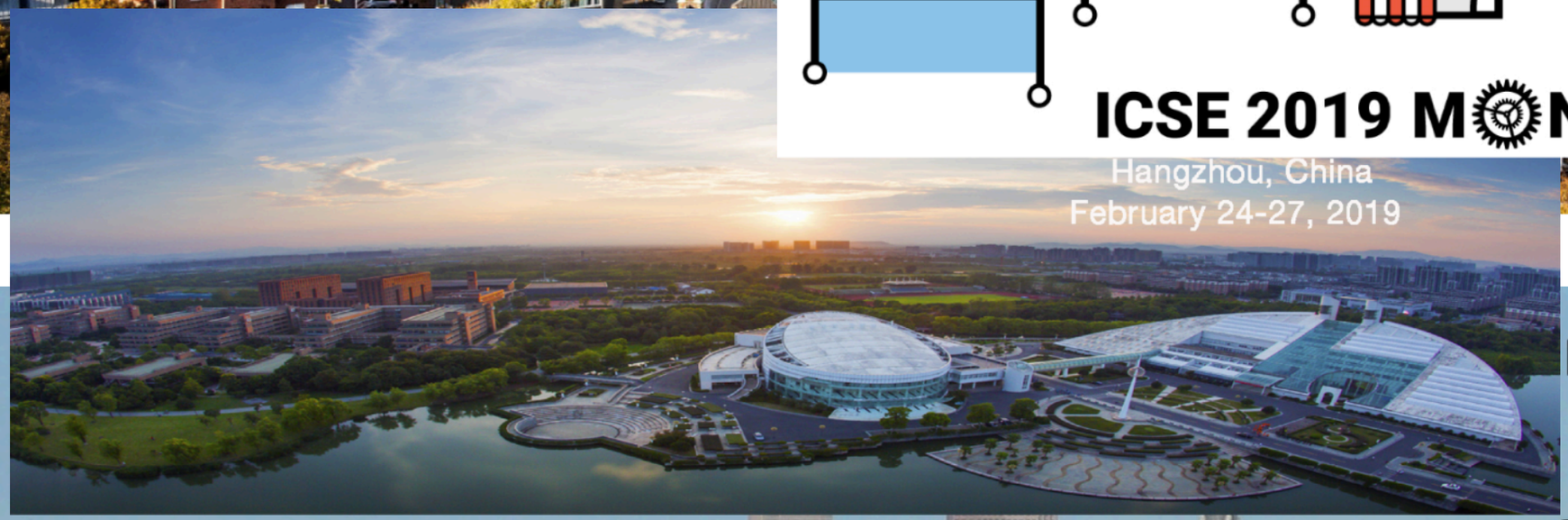
Write a

MSR 2018

Mon 28 - Tue 29 May 2018 Gothenburg, Sweden

co-located with * ICSE 2018 *

Attending | Program | Tracks | Organization | Search | Series | Sign in | Sign up



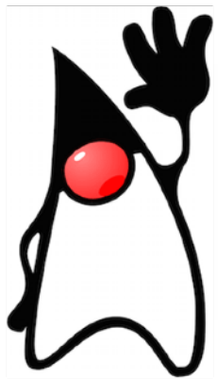
VI Workshop on Software Visualization, Evolution and Maintenance


git


GitHub


stackoverflow


Google Play



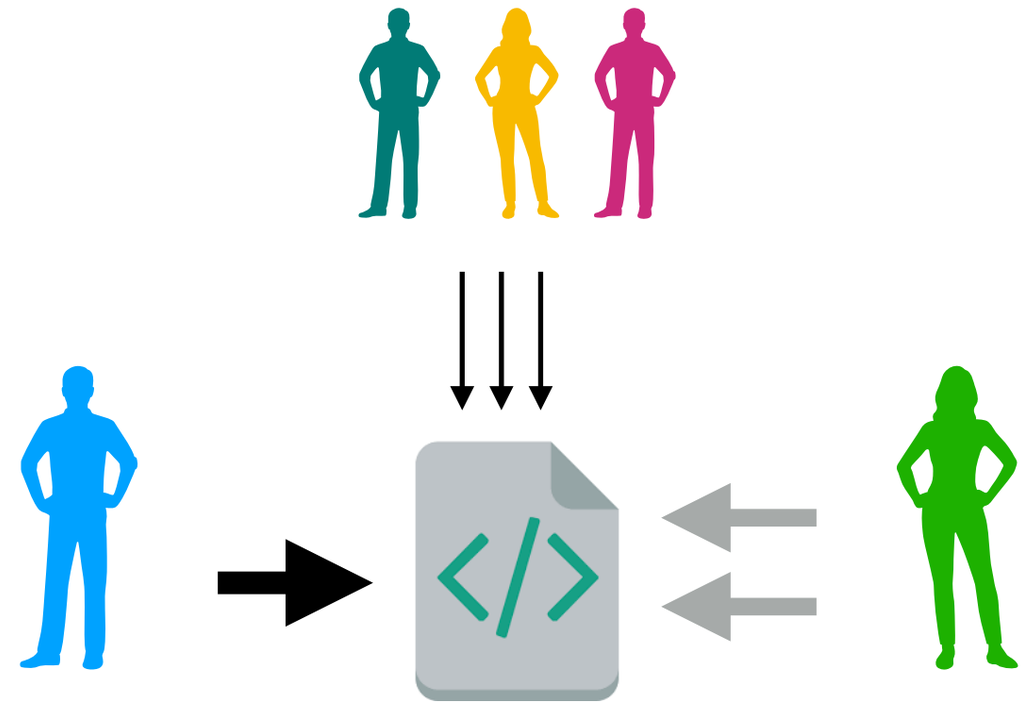

Apache
maven


Jenkins


Travis CI


eclipse

Pesquisas (exemplos)



Who Can Maintain this Code?

Assessing the Effectiveness of Repository-Mining Techniques for Identifying Software Maintainers

Guilherme Avelino[1,2], Leonardo Passos[3], Fabio Petrillo[4], Marco Tulio Valente[1]

[1]Federal University of Minas Gerais, Brazil
[2]Federal University of Piaui, Brazil
[3]Quantstamp Technologies, Canada
[4]Polytechnique Montreal, Canada
{gaa, mtov}@dcc.ufmg.br, leo@quantstamp.com, fabio@petrillo.com

Abstract

In large and complex systems, identifying developers capable of maintaining a piece of code is an important but challenging task. In this context, repository-mining techniques can help by providing some level of automation. Still, whether such techniques effectively

```
        while (!stack.isEmpty()) {

            State current = stack.pop();

//          int theDepth = current.getDepth();
//
//          if(theDepth> maxDepthSearched)
//              maxDepthSearched = theDepth;

            if (current.isGoal(maze)){
                // TODO update the maze if a solution found
                //while(current != null){
                while(current.getParent() != null){
                    if(!current.isGoal(maze))
                        maze.setOneSquare(current.getSquare(), '.');
                    current = current.getParent();

                }
                return true;
            }
```

Commented-out code: Código "removido" através de comentário

Minerando Código Comentado

Lucas Grijó[1], **Andre Hora**[1]

[1] Faculdade de Computação (FACOM)
Universidade Federal de Mato Grosso do Sul (UFMS)

rksgrijo@gmail.com, hora@facom.ufms.br

Abstract. *As software evolves, programmers often comment code snippets as a mean of removing them (a practice known as comment-out code). However, this bad practice may cause problems to software maintainers, such as readability reduction, distraction, and waste of time. In this context, this paper presents an empirical study to evaluate the presence of commented-out code on open source systems. We analyze 100 relevant software systems and 300 releases. As a result, we detect a rate of 4,17% commented-out code. However, we verify that*

```
 package org.jboss.as.modcluster;

-import junit.framework.Assert;
 import org.jboss.as.controller.PathAddress;
 import org.jboss.dmr.ModelNode;
 import org.jboss.dmr.ModelType;
+import org.junit.Assert;
 import org.junit.Test;
```

```
   public Collection<FormulaData> getChildren() {
-    List<FormulaData> result = new ArrayList<FormulaData>();
+    List<FormulaData> result = Lists.newArrayList();
     for (DecoratorContext childContext : decoratorContext.getChildren()) {
       result.add(new DefaultFormulaData(childContext));
     }
```

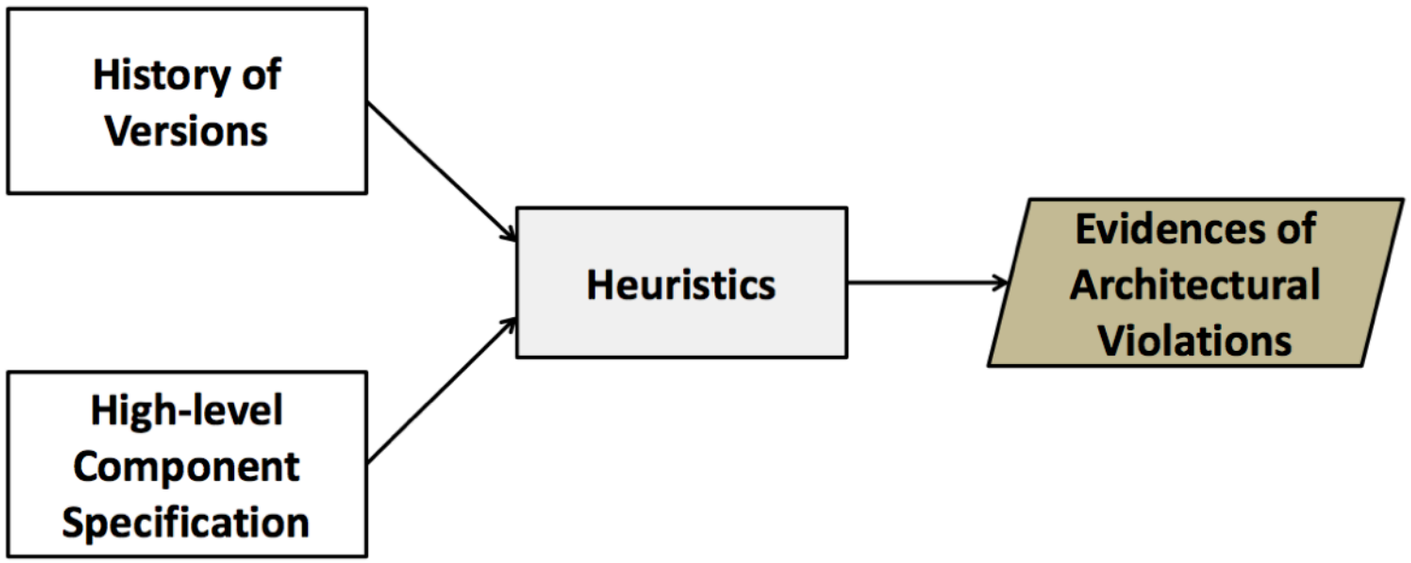
Minerando Mensagens de Depreciação Faltantes em APIs: Um Estudo de Caso no Ecossistema Android

Pedro Henrique de Moraes[1], **Caroline Lima**[1], **Andre Hora**[1]

[1] Faculdade de Computação (FACOM)
Universidade Federal de Mato Grosso do Sul (UFMS)

pedhmoraes@gmail.com, carollimaxp@gmail.com, hora@facom.ufms.br

Abstract. *To facilitate library migration, developers should deprecate APIs before any change that may impact on client systems. However, the literature reports that APIs are commonly deprecated without any message to support their clients. Therefore, in this paper, we propose an approach to recommend missing replacement messages in deprecated APIs. We assess the history version of the Android framework and 100 client systems. We found that some replacement messages can be detected by mining the clients, but alternatives solutions are common. Finally, based on our learning, we suggest improvements to the design of an API deprecation migration tool.*



Mining Architectural Violations from Version History

Cristiano Maffort · Marco Tulio Valente · Ricardo Terra · Mariza Bigonha · Nicolas Anquetil · André Hora

Received: data / Accepted: date

Abstract Software architecture conformance is a key software quality control activity that aims to reveal the progressive gap normally observed between concrete and planned software architectures. However, formally specifying an architecture can be difficult, as it must be done by an expert of the system having a high level understanding of it. In this paper, we present a lightweighted approach for architecture conformance based on a combination of static and historical source code analysis. The proposed approach relies on four heuristics for detecting absences (something expected was

Ref. Type	RDiff Precision	RDiff Recall
Rename Type	1.000	1.000
Move Type	1.000	0.968
Extract Superclass	1.000	0.875
Rename Method	1.000	0.943
Pull Up Method	1.000	0.600
Push Down Method	1.000	0.971
Move Method	1.000	1.000
Extract Method	1.000	0.897
Inline Method	1.000	0.981
Pull Up Field	1.000	0.576
Push Down Field	1.000	0.929
Move Field	1.000	0.269

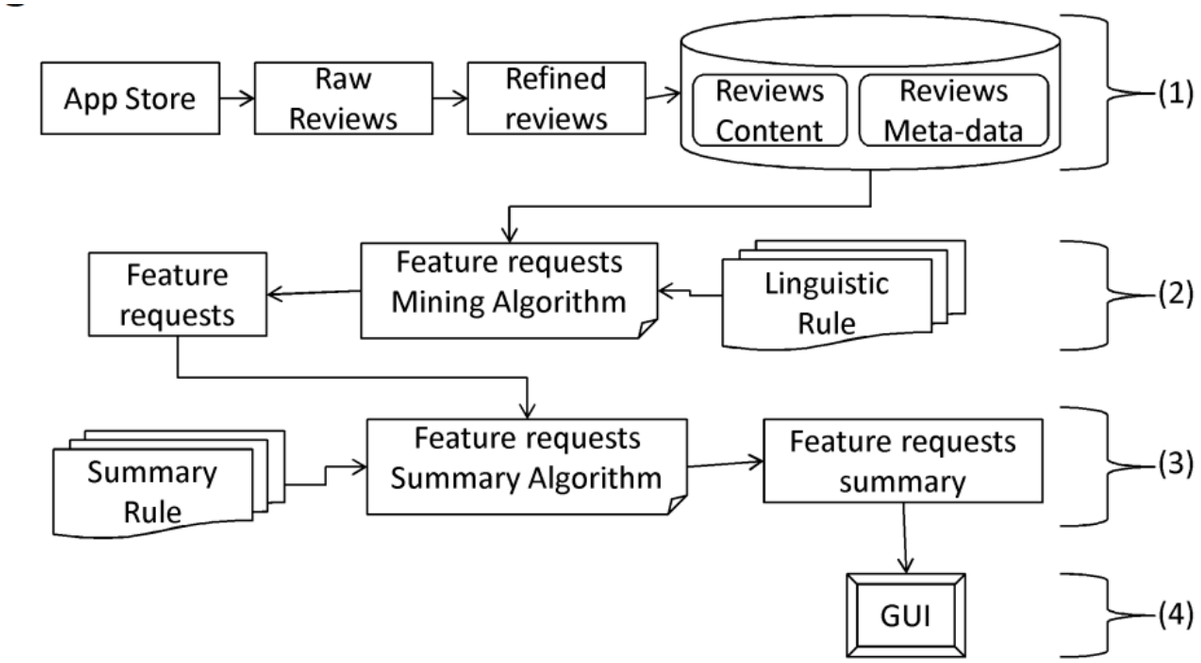
RefDiff: Detecting Refactorings in Version Histories

Danilo Silva*, Marco Tulio Valente†
Department of Computer Science
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
Email: *danilofs@dcc.ufmg.br, †mtov@dcc.ufmg.br

***Abstract*—Refactoring is a well-known technique that is widely adopted by software engineers to improve the design and enable the evolution of a system. Knowing which refactoring operations were applied in a code change is a valuable information to understand software evolution, adapt software components, merge code changes, and other applications. In this paper, we present RefDiff, an automated approach that identifies refactorings performed between two code revisions in a git repository. RefDiff employs a combination of heuristics based on static analysis and code similarity to detect 13 well-known refactoring types. In an**

the refactorings that were applied to an API, we can replay them on the client code automatically.

Although there are approaches capable of detecting refactorings automatically, there are still some issues that hinder their application. Specifically, the precision and recall of such approaches still need improvements. In this paper, we try to fill this gap by proposing RefDiff, an automated approach that identifies refactorings performed in the version history of a system. RefDiff employs a combination of heuristics



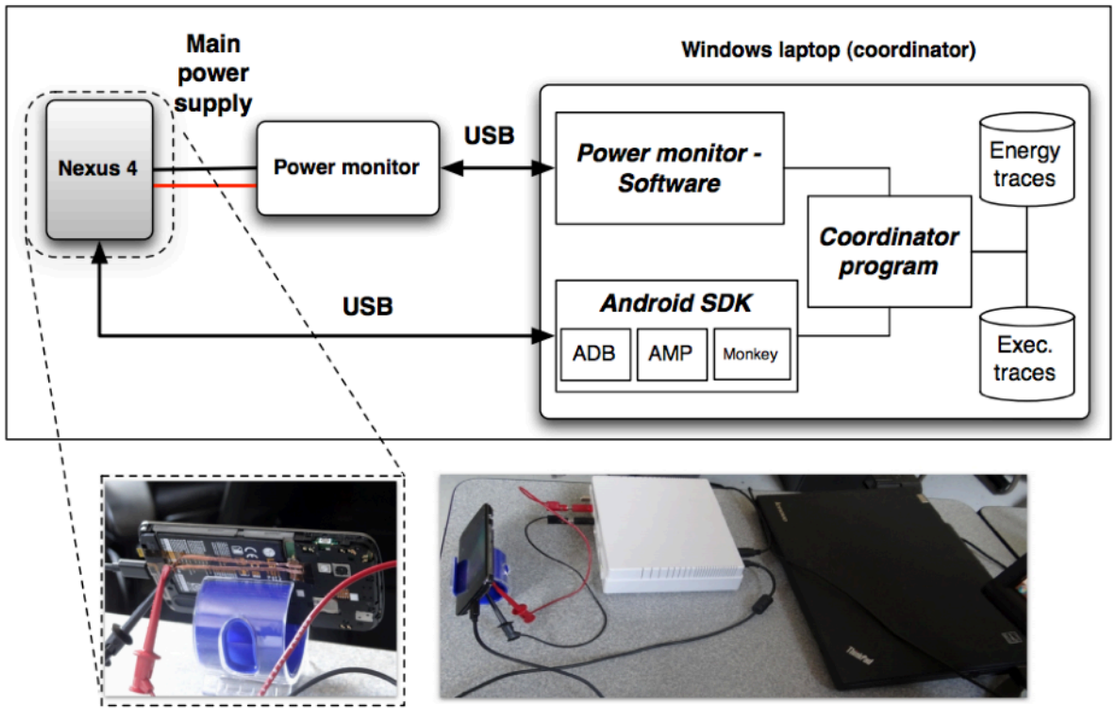
Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews

Claudia Iacob, Rachel Harrison
Department of Computing and Communication Technologies
Oxford Brookes University
Oxford, United Kingdom
{iacob, rachel.harrison}@brookes.ac.uk

***Abstract*—Mobile app reviews are valuable repositories of ideas coming directly from app users. Such ideas span various topics, and in this paper we show that 23.3% of them represent feature requests, i.e. comments through which users either suggest new features for an app or express preferences for the re-design of already existing features of an app. One of the challenges app developers face when trying to make use of such feedback is the massive amount of available reviews. This makes it difficult to identify specific topics and recurring trends across reviews. Through this work, we aim to support such processes by designing MARA (Mobile App Review Analyzer), a prototype for automatic retrieval of mobile app feature requests from online reviews. The design of the prototype is a) informed by an investigation of the ways users express feature requests through reviews, b) developed around a set of pre-defined linguistic rules, and c) evaluated on a large sample of online reviews. The results**

Facebook app alone has almost 6 million reviews. Moreover, the most popular non free apps on the Google App store get an average of 45.5 reviews per day. Second, reviews have a style of their own; they are usually short, unstructured and seldom obey grammar and punctuation rules. Moreover, users express opinions in unconventional ways, often using unconventional syntax or sarcasm to vent their feelings after purchasing an app. Even though humans can usually interpret such reviews, it is challenging to automate such interpretations [6].

We first considered a sample of reviews for analysis and we aimed to identify how much of the users' feedback consists of feature requests and how these feature requests are expressed through reviews. Such findings informed the design of a prototype system able to identify and retrieve feature



Mining Energy-Greedy API Usage Patterns in Android Apps: An Empirical Study

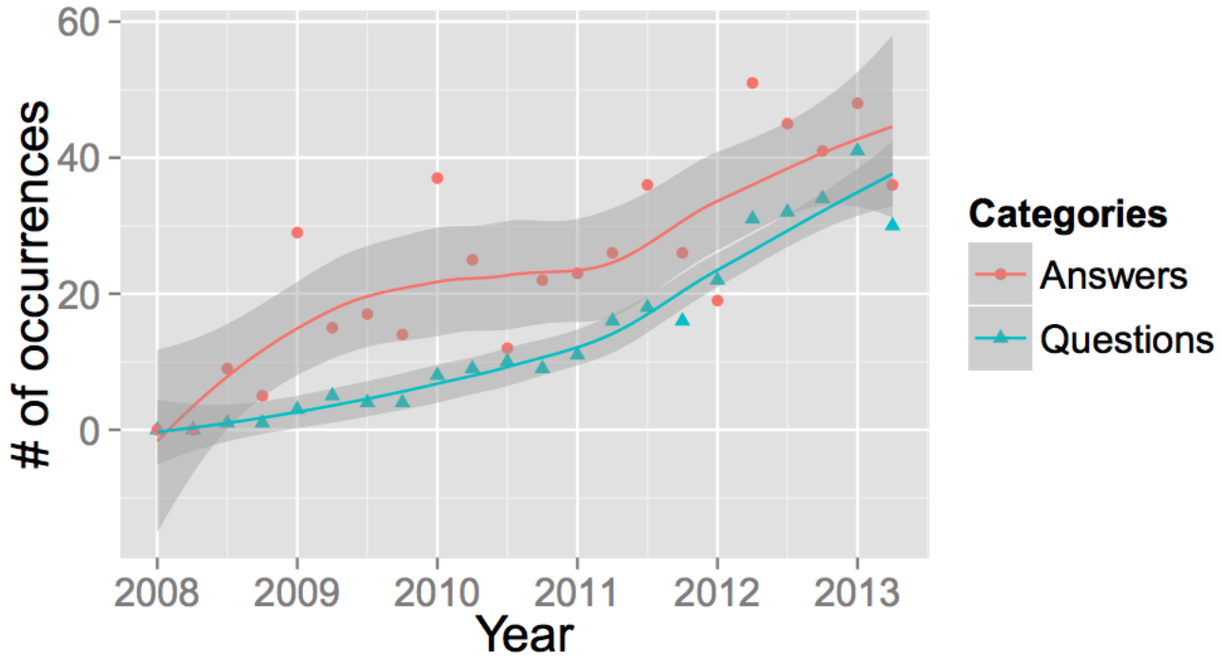
Mario Linares-Vásquez[1], Gabriele Bavota[2], Carlos Bernal-Cárdenas[3]
Rocco Oliveto[4], Massimiliano Di Penta[2], Denys Poshyvanyk[1]
[1]The College of William and Mary, Williamsburg, VA, USA [2]University of Sannio, Benevento, Italy
[3]Universidad Nacional de Colombia, Bogotá, Colombia [4]University of Molise, Pesche (IS), Italy
mlinarev@cs.wm.edu, gbavota@unisannio.it, cebernalc@unal.edu.co,
rocco.oliveto@unimol.it, dipenta@unisannio.it, denys@cs.wm.edu

ABSTRACT

Energy consumption of mobile applications is nowadays a hot topic, given the widespread use of mobile devices. The high demand for features and improved user experience, given the available powerful hardware, tend to increase the apps' energy consumption. However, excessive energy consumption in mobile apps could also be a consequence of energy greedy hardware, bad programming practices, or particular API usage patterns. We present the largest to date quantitative and qualitative empirical investigation into the categories of API calls and usage patterns that—in the context of the

1. INTRODUCTION

In recent years, we are observing rapid evolution of mobile devices in terms of available hardware, operating systems, and, as a consequence of that, the growing lists of features that mobile applications' (apps) users demand. These modern apps have virtually the same features as their equivalent desktop applications. For instance, many top video games for mobile devices provide similar levels of user experience as compared to those console analogs. Such evident step-ahead has, however, a price to be paid. Nowadays, multi-core processors, high-performance Graphical Processing Units



Mining Questions about Software Energy Consumption

Gustavo Pinto[†‡], Fernando Castor[†], Yu David Liu[‡]

[†]Federal University of Pernambuco
Recife, PE, Brazil
{ghlp, castor}@cin.ufpe.br

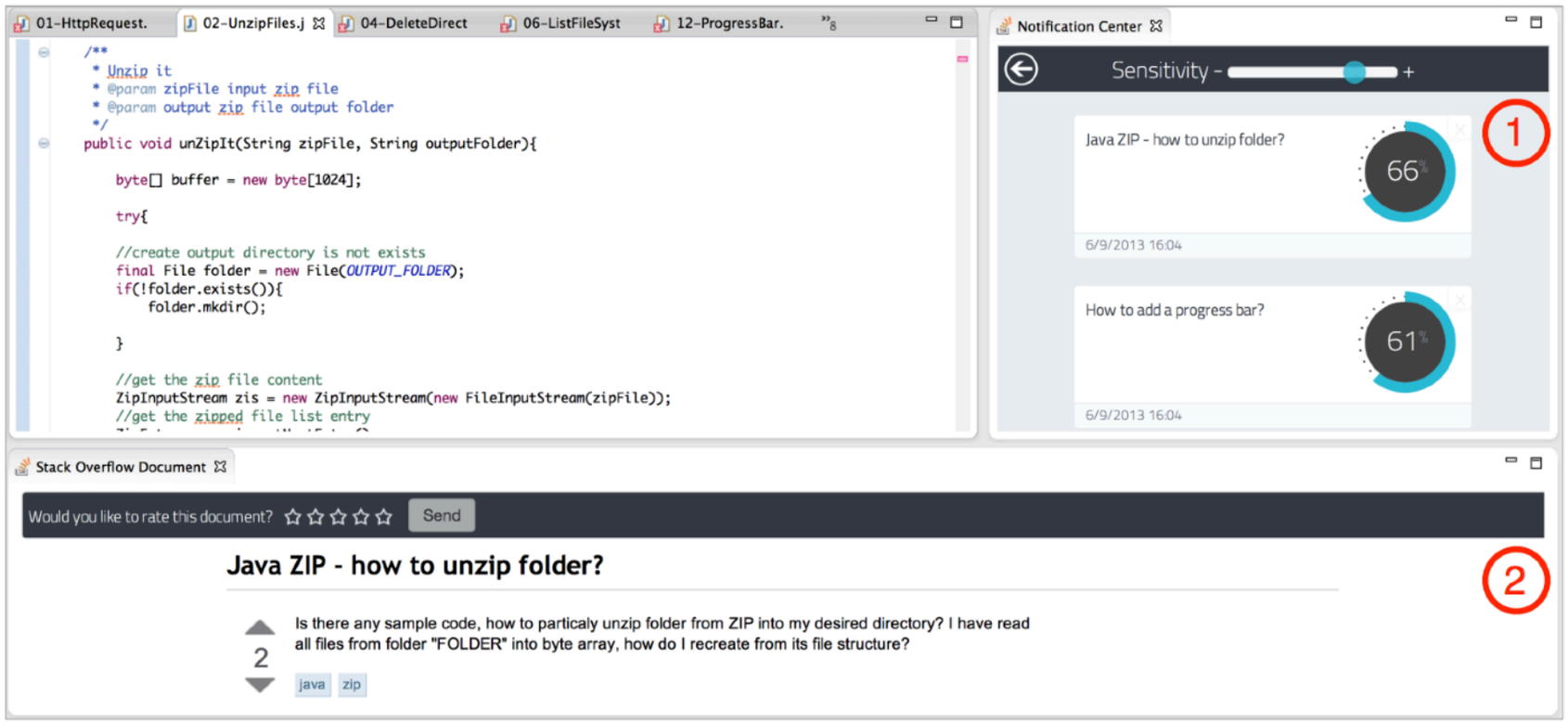
[‡]SUNY Binghamton
Binghamton, NY 13902, USA
davidL@cs.binghamton.edu

ABSTRACT

A growing number of software solutions have been proposed to address application-level energy consumption problems in the last few years. However, little is known about how much software developers are concerned about energy consumption, what aspects of energy consumption they consider important, and what solutions they have in mind for improving energy efficiency. In this paper we present the first empirical study on understanding the views of application programmers on software energy consumption problems. Using StackOverflow as our primary data source, we analyze a carefully curated sample of more than 300 questions and 550 answers from more than 800 users. With this data, we

1. INTRODUCTION

Nowadays, thanks to the rapid proliferation of mobile phones, tablets, and unwired devices in general, energy efficiency is becoming a key software design consideration where the energy consumption is closely related to battery lifetime. It is also of increasing interest in the non-mobile arena, such as data centers and desktop environments. Energy-efficient solutions are highly sought after across the compute stack, with more established results through innovations in hardware/architecture [2, 14, 28], operating systems [10, 19, 24], and runtime systems [8, 25, 31]. In recent years, there is a growing interest in studying energy consumption from higher layers of the compute stack and most of these stud-



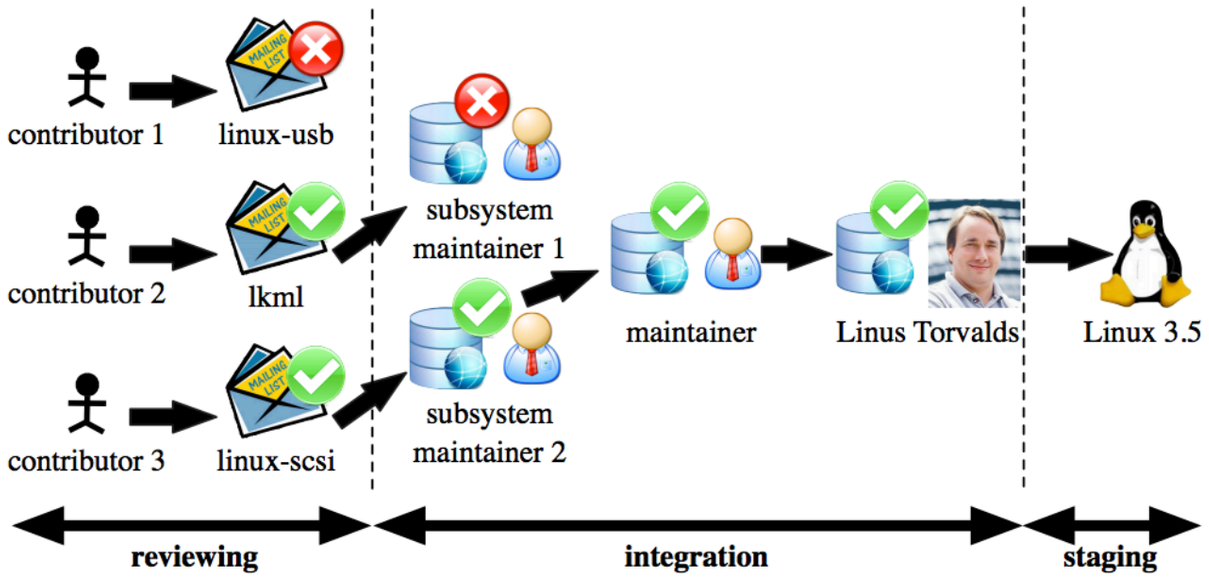
Mining StackOverflow to Turn the IDE into a Self-Confident Programming Prompter

Luca Ponzanelli[1], Gabriele Bavota[2], Massimiliano Di Penta[2], Rocco Oliveto[3], Michele Lanza[1]
1: REVEAL @ Faculty of Informatics – University of Lugano, Switzerland
2: University of Sannio, Benevento, Italy 3: University of Molise, Pesche (IS), Italy

ABSTRACT

Developers often require knowledge beyond the one they possess, which often boils down to consulting sources of information like Application Programming Interfaces (API) documentation, forums, Q&A websites, *etc.* Knowing what to search for and how is non-trivial, and developers spend time and energy to formulate their problems as queries and to peruse and process the results.

We propose a novel approach that, given a context in the IDE, problems, the main one being the absence of automation: Every time developers need to look for information, they interrupt their work flow, leave the IDE, and use a Web browser to perform and refine searches, and assess the results. Finally, they transfer the obtained knowledge to the problem context in the IDE. The information is retrieved from different sources, such as forums, mailing lists [2], blogs, Q&A websites, bug trackers [1], *etc.* A prominent example is Stack Overflow, popular among developers as a venue for sharing



Will My Patch Make It? And How Fast?

Case Study on the Linux Kernel

Yujuan Jiang, Bram Adams
MCIS, Polytechnique Montréal, Canada
{yujuan.jiang,bram.adams}@polymtl.ca

Daniel M. German
University of Victoria, Canada
dmg@uvic.ca

***Abstract*—The Linux kernel follows an extremely distributed reviewing and integration process supported by 130 developer mailing lists and a hierarchy of dozens of Git repositories for version control. Since not every patch can make it and of those that do, some patches require a lot more reviewing and integration effort than others, developers, reviewers and integrators need support for estimating which patches are worthwhile to spend effort on and which ones do not stand a chance. This paper cross-links and analyzes eight years of patch reviews from the kernel mailing lists and committed patches from the Git repository to understand which patches are accepted and how long it takes those patches to get to the end user. We found that 33% of the patches makes it into a Linux release, and that most of them need 3 to 6 months for this. Furthermore, that patches developed by more experienced developers are more easily accepted and faster reviewed and integrated. Additionally, reviewing time is impacted**

ones, whose changes did not make it after putting months of work into them (e.g., [12], [13]). Even major projects like the Google Android mobile platform have problems getting their Linux kernel modifications integrated into the official kernel version [11]. Yet, determining up front whether a patch will make it, and how long it will take, is a grey area. Research on code reviews has shown how small patches [6], [4] sent by experienced developers [2] are more likely to be accepted by the reviewers, but it is not clear if these characteristics play the same role during the actual integration of the patch with other patches. Similarly, the impact of these characteristics on the time it takes to get a patch into a release is unclear.

This paper studies the relation of patch characteristics with