Achievements and Challenges in Software Reverse Engineering

Gerard o Canfora, Massimiliano Di Penta,
and Luigi Cerulo (CACM, April 2011)

Introduction

- After the pioneering work of Lehman we know that real-world software systems require continuous change and enhancement

- The need for modifying software induces the need to comprehend it.

- Software comprehension is challenged by the lack of adequate and up-to-date software documentation

- Software reverse engineering is
  - “the process of analyzing a subject system to identify the system’s components and their inter-relationships and create representations of the system in another form or at a higher level of abstraction”

Introduction

- Reverse engineering is a process of examination rather than a process of change

- The core of reverse engineering is deriving, from available software artifacts, representations understandable by humans

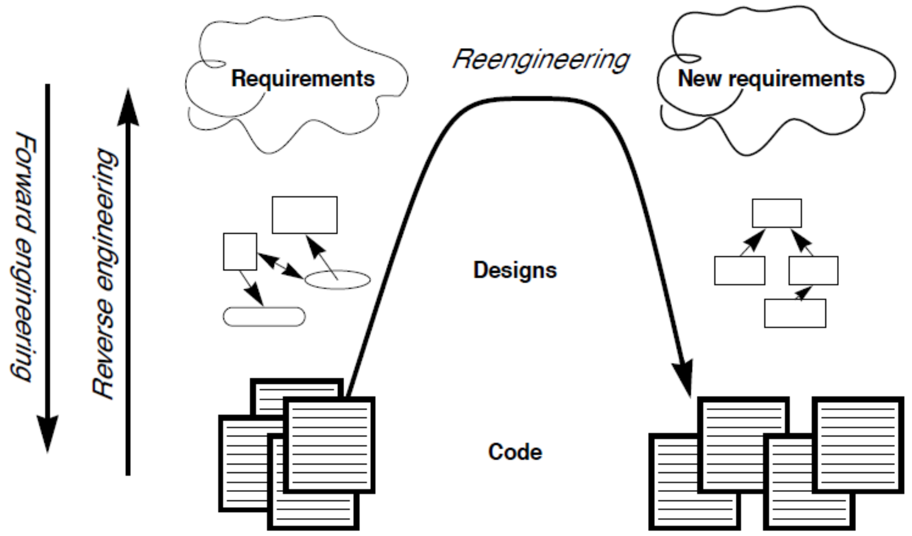


Figure 1.1: Forward, reverse and reengineering

Reverse Engineering Concepts

- A reverse engineering activity is performed by a software engineer to solve some specific problems related to a software product, consisting of software artifacts
  - For example, source code, executables, project documentation, test cases, and change requests

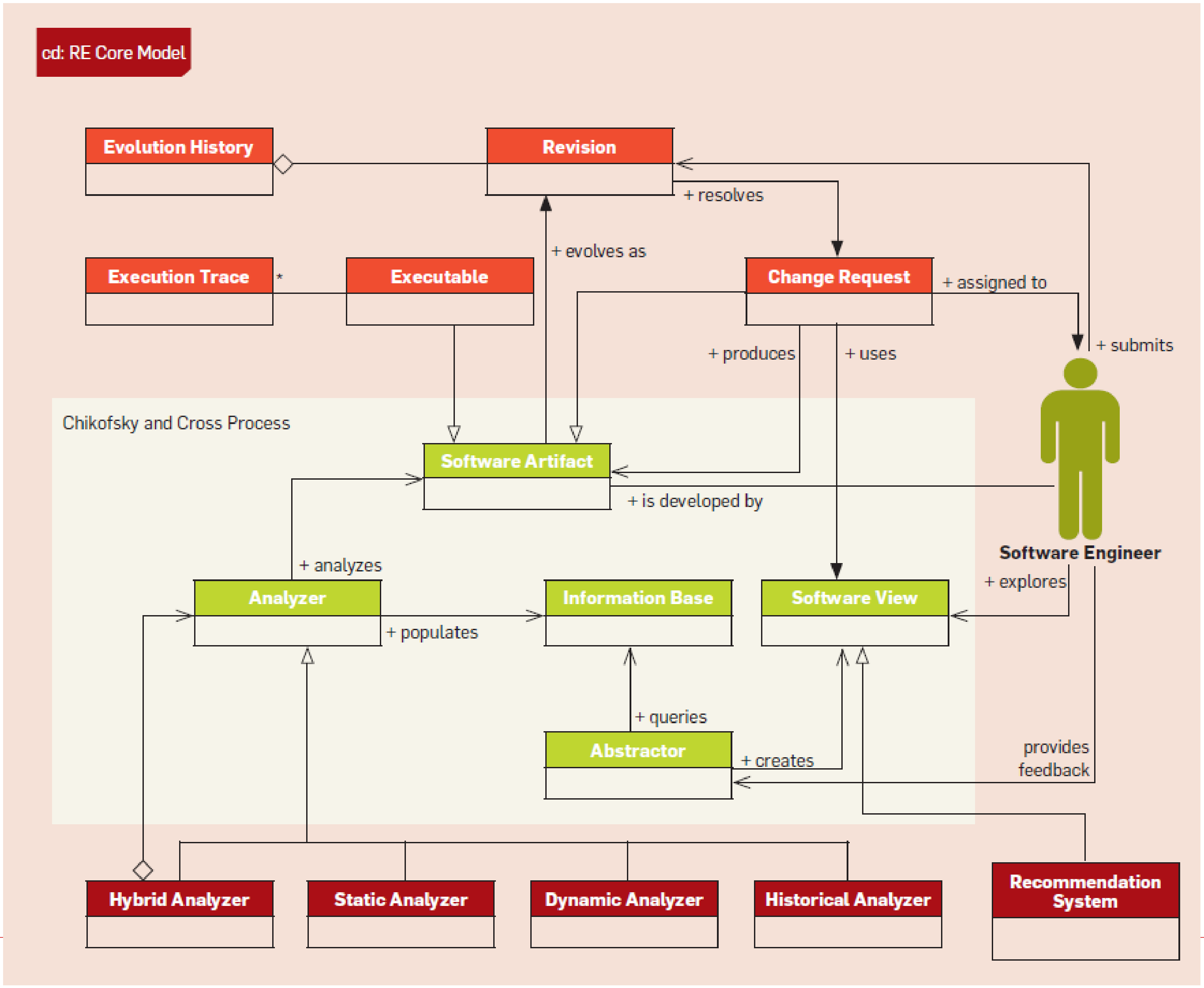
- Software engineers benefit of reverse engineering by exploring software artifacts by means of software views, representations, sometimes visual, aimed at increasing the current comprehension of a software product, or at favoring maintenance and evolution activities.

Reverse Engineering Concepts

- Software views are built by means of an **abstractor**, which in turn uses information extracted from software artifacts and stored in the information base by an **analyzer**.

- Often reverse engineering aims at analyzing the evolution of software artifacts across their revisions, which occur to satisfy change requests.

Reverse Engineering Concepts



Reverse Engineering Concepts

- The **analyzers** can extract information by means of
  - Static analysis (static analyzers)
  - Dynamic analysis (Dynamic Analyzers)
  - Combination of the two (hybrid analyzers).
- Recently, historical analyzers, which extract information from the evolution repository of a software product, are gaining popularity.
- The software engineer can provide feedbacks to the reverse engineering tool to produce refined and more precise views.
- A particular type of software view that emerged recently are **recommendation systems**, which provide suggestions to the software engineer and, if needed, trigger a new change request

Software Analysis

- Software analysis is performed by analyzers—tools that take software artifacts as input and extract information relevant to reverse engineering tasks.

- Software analysis can be:
  - **Static**, when it is performed, within a single system snapshot, on software artifacts without requiring their execution;
  - **Dynamic**, when it is performed by analyzing execution traces obtained from the execution of instrumented versions of a program, or by using an execution environment able to capture facts from program executions;
  - **Historical**, when the aim is to gain information about the evolution of the system under analysis by considering the changes performed by developers to software artifacts, as recorded by versioning systems.

Static Analyzers

- Static analyzers must deal with different language variants and non-compilable code.

- Static analysis is reasonably fast, precise, and cheap.

- However, many peculiarities of programming languages, such as pointers and polymorphism, or dynamic class loading, make static analysis difficult and sometimes **imprecise**

Dynamic Analyzers

- To overcome the limitations of static analysis, reverse engineers can count on dynamic analysis, which extracts information from execution traces.
- However, dynamic analysis can be **incomplete**, because it depends on program inputs
- Challenge: ability to mine relevant information from execution traces.
  - Execution traces tend to quickly become large and unmanageable, thus a relevant challenge is to filter them and extract information relevant for the particular understanding task being performed

On-the-fly extraction of hierarchical object graphs

Hugo de Brito; Humberto Torres Marques-Neto; Ricardo Terra; Henrique Rocha; Marco Tulio Valente,

Journal of the Brazilian Computer Society, To appear

Introdução

- Arquitetura de software se preocupa em descrever:
  - Principais componentes de um sistema
  - Restringir relacionamentos entre tais componentes

- Problema:
  - Muitos sistemas não possuem documentação arquitetural
  - Quando possuem, documentação pode estar desatualizada

- Possível “solução”: Engenharia Reversa
- Duas técnicas:
  - Análise Estática
  - Análise Dinâmica

Análise Estática

- Diagramas: Classe, Pacote, Matrizes Dependência etc
- Vantagem: não demandam execução
- Desvantagem #1:
  - Informações sobre “ordem” dos eventos é perdida
  - Por onde começo a ler um diagrama?
- Desvantagem #2:
  - Relacionamentos dinâmicos não são capturados
  - Exemplo: polimorfismo, chamada dinâmica, reflexão etc

Análise Dinâmica

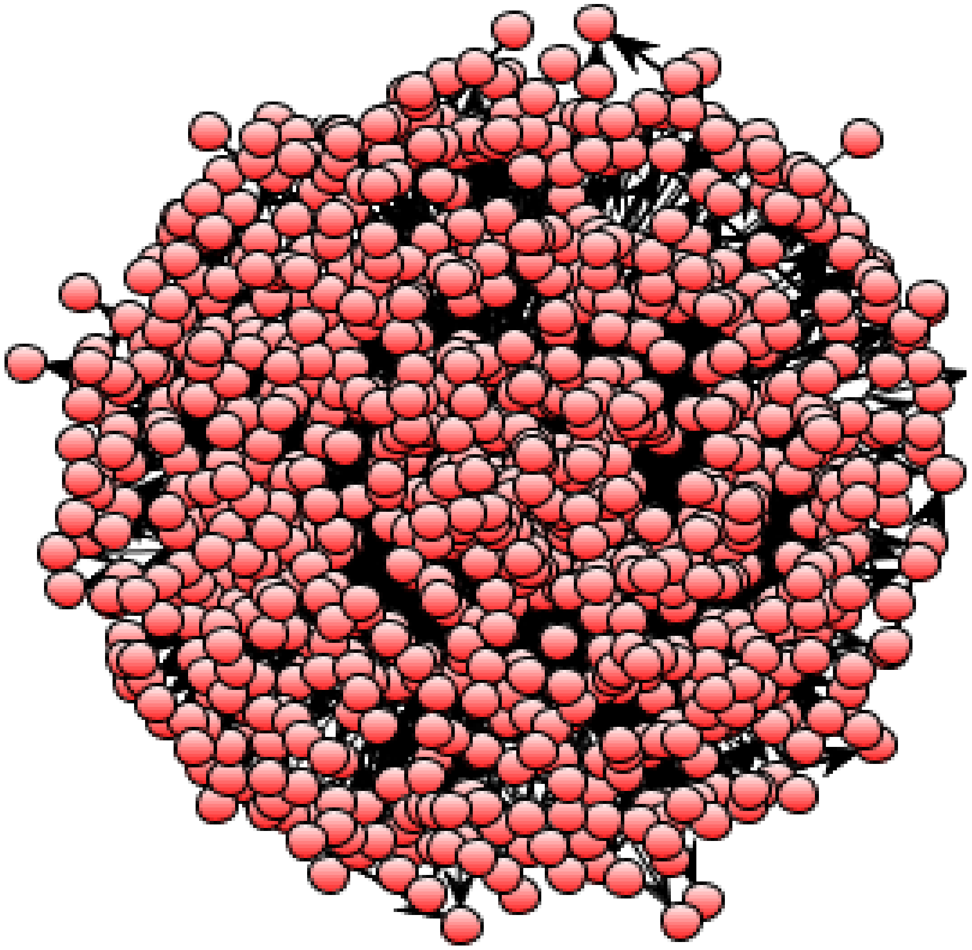
- Diagramas de Objeto, Colaboração, Sequência etc

- Vantagem #1:
  - Mais fácil acompanhar o que diagramas revelam
  - Exibem o fluxo de execução do sistema

- Desvantagem #1:
  - Demandam execução do sistema

- Desvantagem #2:
  - Escalabilidade
  - Crescimento explosivo do número de elementos dos diagramas

Diagrama de Objetos (**JHotDraw**)



Escalabilidade: Soluções Existentes

- Agrupar objetos em unidades de maior granularidade
  - Clusters, Componentes, Domínios, Sistemas etc

- Duas abordagens: automática ou manual

- Abordagens automáticas:
  - Usando algoritmos de clusterização
  - Problema: resultados não são aqueles esperados

- Abordagens manuais:
  - Usando anotações no código. Ex.: **`@View`** `class A {...}`
  - Problema: invasivo, trabalhoso (para sistema legados)

Proposta: Object Graphs (OG)

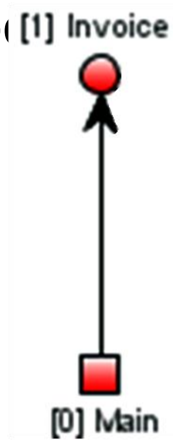
- Suporte a grupos de objetos de maior granularidade
  - Chamados domínios
  - Definidos de forma não-invasiva, via ling. de expressões regulares

- Suporte aos vários relacionamentos que existem em OO
  - Chamada dinâmica de métodos, reflexão computacional, relacionamentos entre objetos e campos estáticos de classes etc

Object Graphs

- Dois tipos de vértices:
  - Círculo: **objeto**
  - Quadrado: **classe** (que acessam objetos via métodos estáticos)

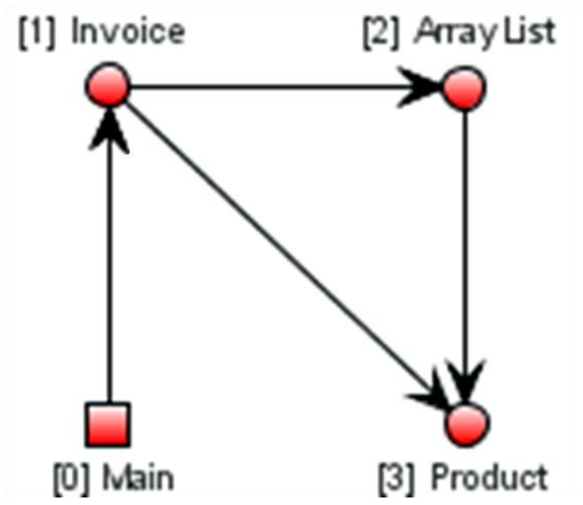
- Nome de um vértice: três campos
  - Inteiro sequencial (indica ordem de criação dos vértices)
  - Classe
  - Cor (objetos criados p sma *thread* tem a mesma cor)



Exemplo #1

```
class Main {
  static Invoice invoice;
   public static void main(…) {
     invoice = new Invoice();
     invoice.load();
   }
}

class Invoice {
  Collection<Product> col;
  void load() {
    col = new ArrayList<Product>();
    Product p = new Product();
    col.add(p);
  }
}
```



Exemplo #2: Domínios

- Domínio: conjunto de classes
  - Recurso para agrupamento de vértices
  - Objetos de um domínio são representados por um único vértice
  - Definidos em um arquivo separado (logo, não invasivos)
  - Usando uma linguagem de expressões regulares
  - Hexágono: vértices que representam domínios

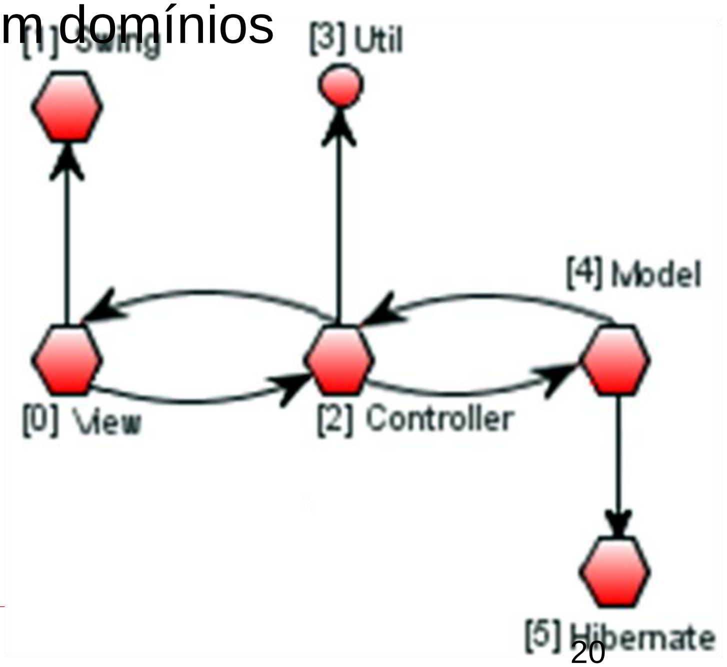
```
domain View: myapp.view.IView+

domain Controller: myapp.controller.*

domain Swing: javax.swing.**

domain Hibernate: org.hibernate.**

domain Model:
  "myapp.model.[a-zA-Z0-9/.]*DAO"
```



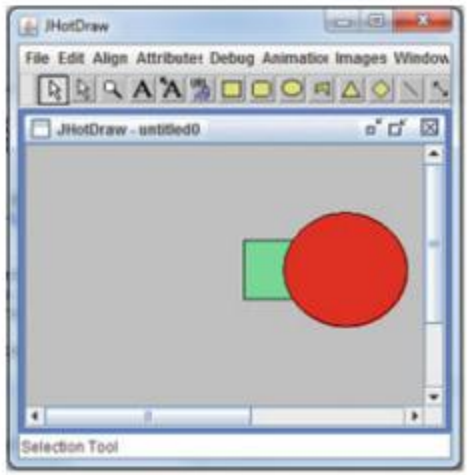
OG: Ferramenta de Visualização

- Principais características:
  - Pode ser "acoplada" a um sistema de forma não invasiva
  - Permite visualização à medida que o programa é executado

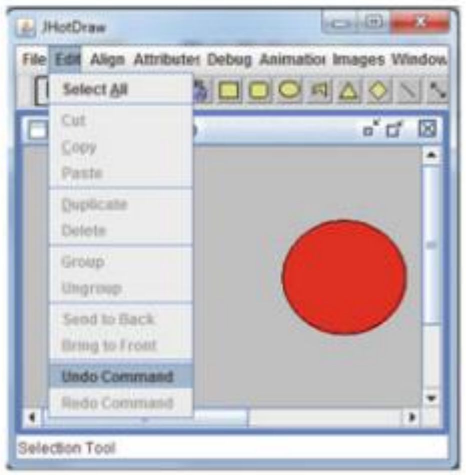
- Instrumentação do sistema alvo é feita via aspectos

Case study: corrective maintenance

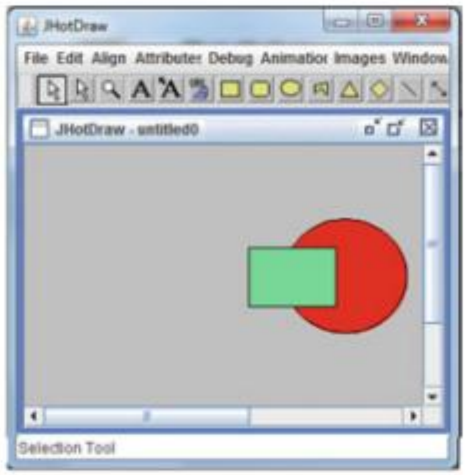
- Bug 1850703 (Opened 2007-12-14): “Redoing Figure delete change order”



(a) Original drawing

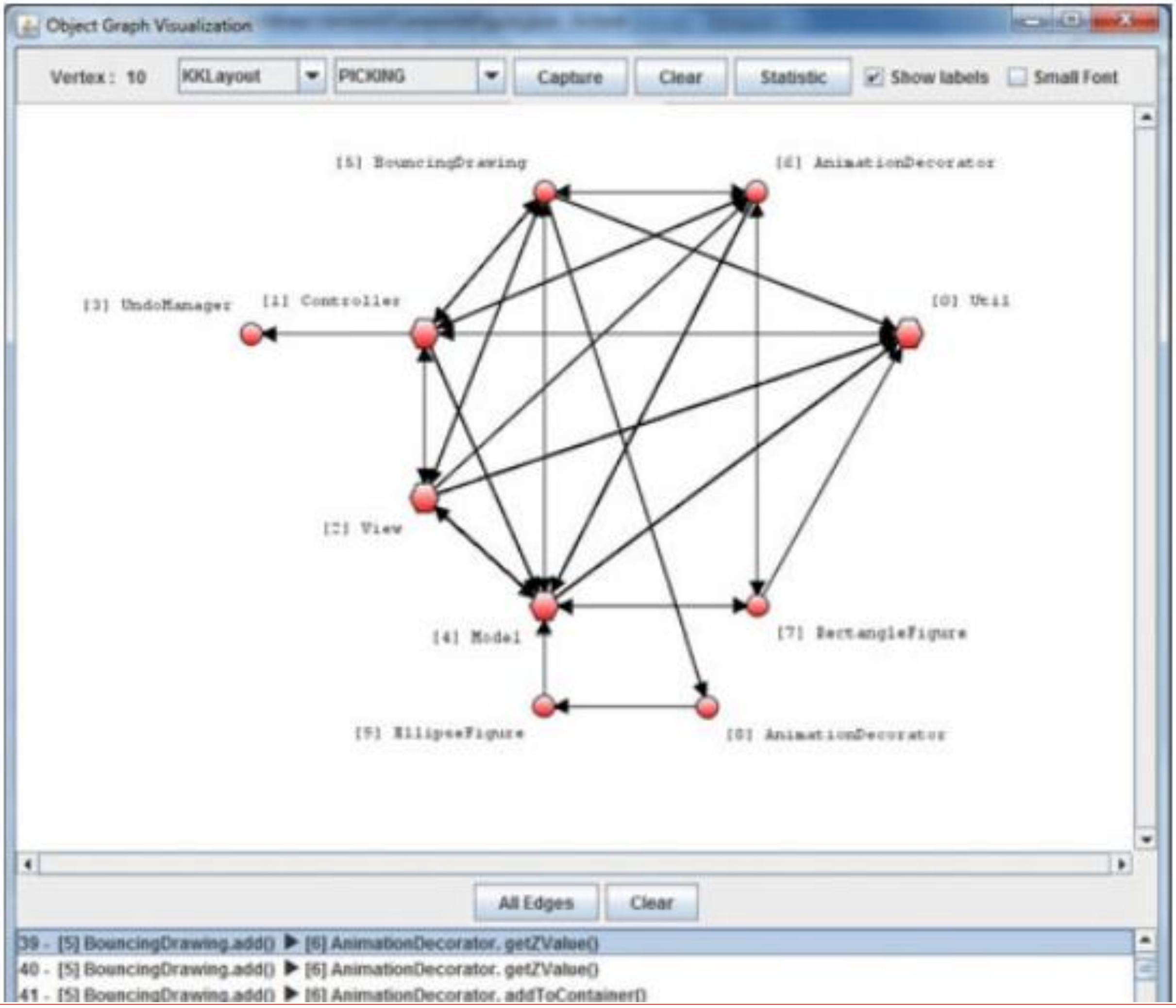


(b) Rectangle deleted



(c) Rectangle restored on top

OG for Bug 1850703



Vertex : 10 KKLayout PICKING [✓ Show labels] Small Font

39 - [5] BouncingDrawing.add() ▶ [6] AnimationDecorator. getZValue()
40 - [5] BouncingDrawing.add() ▶ [6] AnimationDecorator. getZValue()
41 - [5] BouncingDrawing.add() ▶ [6] AnimationDecorator. addToContainer()

OG for Bug 1850703

- We sequentially inspected the OG’s edges to locate possible methods related to the “depth” of a figure in a drawing.

- A call to the method AnimationDecorator.getZValue() (node 6) coming from a BouncingDraw object (node 5) called our attention (since the suffix Zvalue reminds the depth of a figure in the current drawing).

- In fact, by retrieving JHotDraw’s code where this bug has been fixed, it was possible to assert that the changes have been confined to the method BouncingDraw.add(), which was calling getZValue() in an incorrect way

Achievements and Challenges in Software Reverse Engineering

Marco Túlio Valente

DCC - UFMG

Historical Analyzers

- The growing diffusion of versioning systems, bug tracking systems, and other software repositories, such as mailing lists or security advisories, poses the basis for a third dimension of software analysis, namely the historical analysis of data extracted from software repositories

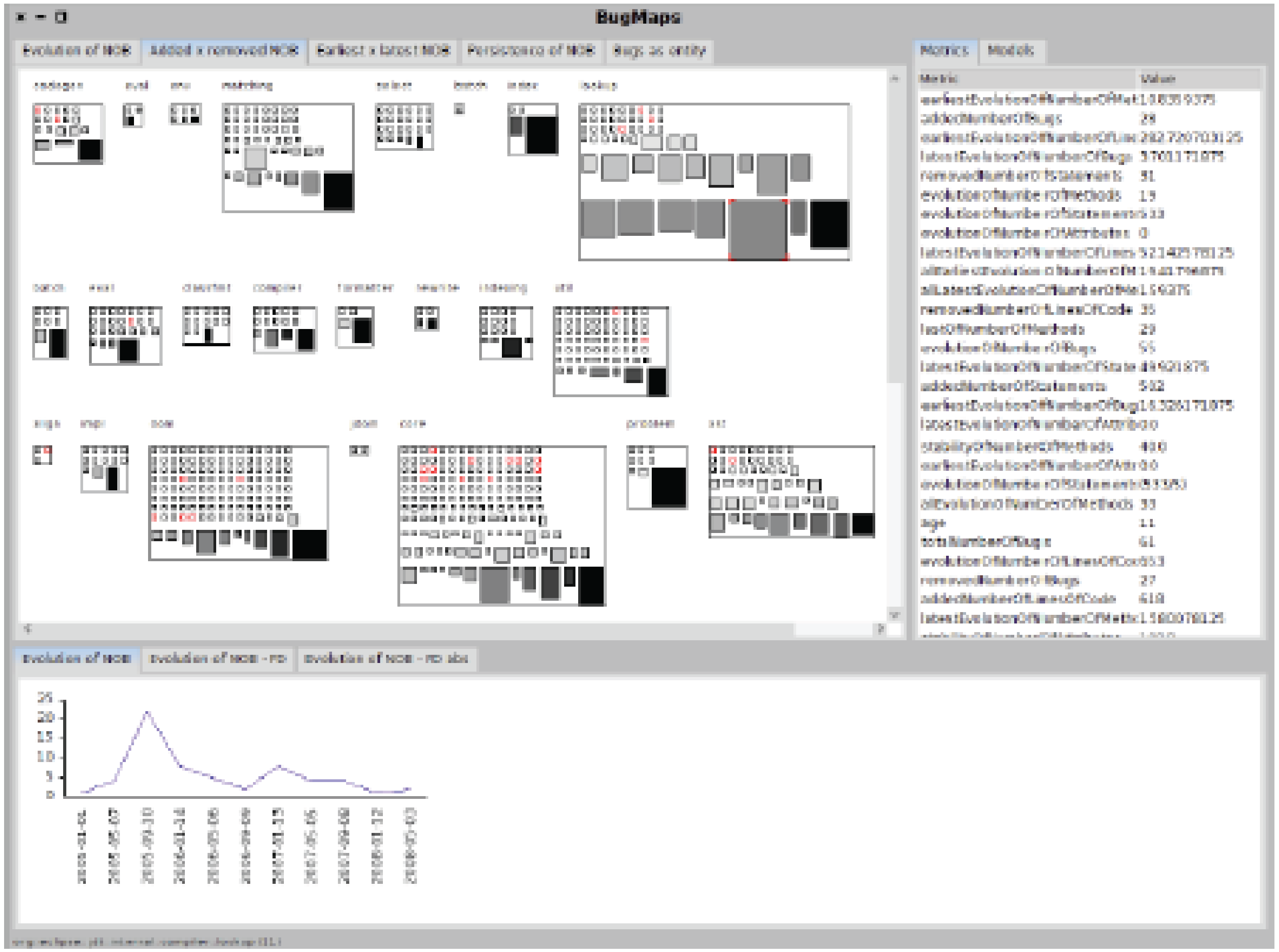
- Challenge:
  - Ability to classify and combine information from different, heterogeneous software repositories.
  - For example, integrating different repositories requires linking a change committed in a versioning system with an issue posted on a bug-tracking system.

BugMaps: A Tool for the Visual Exploration and Analysis of Bugs.

Andre Hora; Nicolas Anquetil; Stephane Ducasse; Muhammad Bhatti; Cesar Couto; Marco Tulio Valente; Julio Martins.

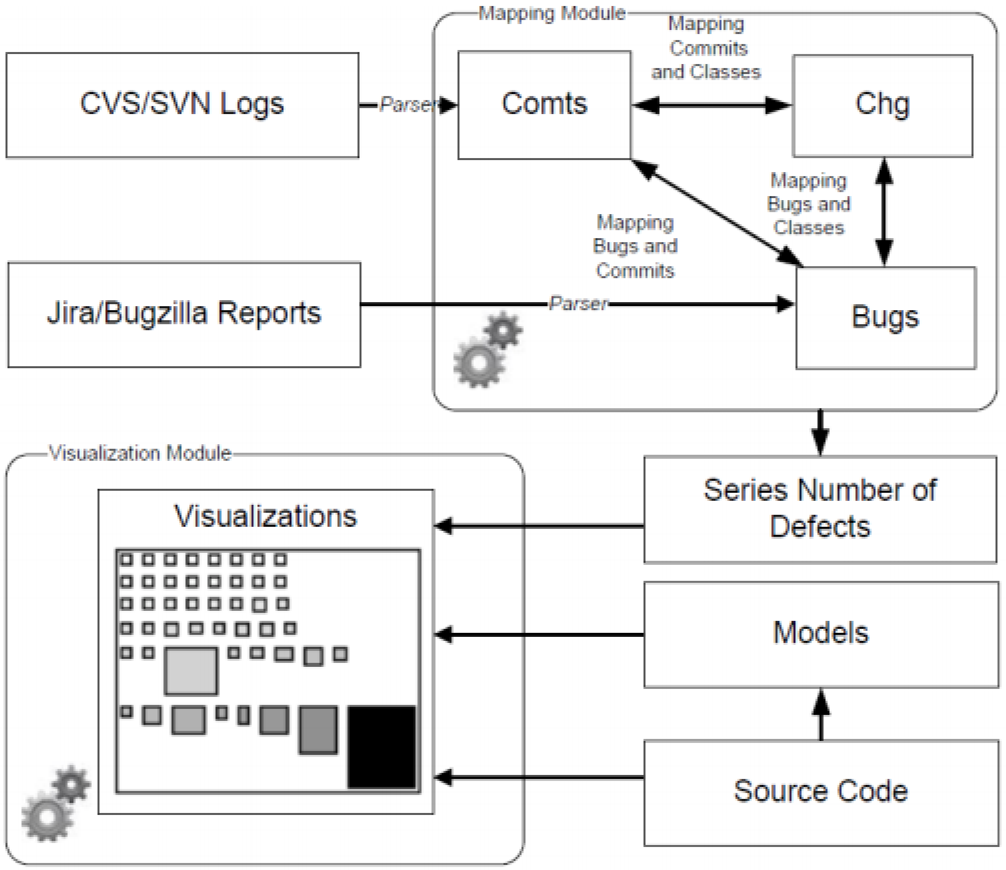
CSMR 2012

Visualization



Metric	Value
earliestEvolutionOfNumberOfMet	1.18359375
addedNumberOfBugs	28
earliestEvolutionOfNumberOfLine	282.720703125
latestEvolutionOfNumberOfBugs	3.701171875
removedNumberOfStatements	31
evolutionOfNumberOfMethods	19
evolutionOfNumberOfStatements	533
evolutionOfNumberOfAttributes	0
latestEvolutionOfNumberOfLines	321.42578125
allEarliestEvolutionOfNumberOfM	1.541796875
allLatestEvolutionOfNumberOfMe	1.59375
removedNumberOfLinesOfCode	35
lastOfNumberOfMethods	29
evolutionOfNumberOfBugs	55
latestEvolutionOfNumberOfState	49.921875
addedNumberOfStatements	532
earliestEvolutionOfNumberOfBug	16.326171875
latestEvolutionOfNumberOfAttrib	0.0
stabilityOfNumberOfMethods	40.0
earliestEvolutionOfNumberOfAttr	0.0
evolutionOfNumberOfStatements	533.53
allEvolutionOfNumberOfMethods	33
age	11
totalNumberOfBugs	61
evolutionOfNumberOfLinesOfCode	553
removedNumberOfBugs	27
addedNumberOfLinesOfCode	610
latestEvolutionOfNumberOfMetho	1.580078125

Architecture



Mapping Module
CVS/SVN Logs
Parser
Comts
Mapping Commits and Classes
Chg
Mapping Bugs and Classes
Mapping Bugs and Commits
Jira/Bugzilla Reports
Parser
Bugs
Visualization Module
Visualizations
Series Number of Defects
Models
Source Code