Achievements and Challenges in Software Reverse Engineering

Gerardo 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.
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.
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.
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)