



Automated runtime software repair

Benoit Cornu, Martin Monperrus

► To cite this version:

Benoit Cornu, Martin Monperrus. Automated runtime software repair. GDR GPL 2013, Apr 2013, Nancy, France. hal-00820076

HAL Id: hal-00820076

<https://hal.inria.fr/hal-00820076>

Submitted on 3 May 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Automated Runtime Software Repair

Improving resilience in presence of exceptions

Benoit Cornu, Martin Monperrus

Context

The server encountered an internal error ...

- > Observation: many applications crash due to unexpected exceptions
- > Goal: catch those exceptions* in an automated manner with synthesized catch blocks to prevent crashing the application
- > Techniques: code transformation, empirical analysis, code synthesis, data-mining

* whether checked or unchecked/runtime exceptions

```

HTTP Status 500 -
Exception report
message The server encountered an internal error () that prevented it from fulfilling the request.
exception
javax.servlet.ServletException: Servlet execution threw an exception
root cause
java.lang.ClassNotFoundException: org.apache.commons.io.output.DeferredFileOutputStream
org.apache.commons.io.output.DeferredFileOutputStream: java:103
org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader: java:1209)
java.lang.ClassLoader.loadClassInternal(ClassLoader: java:310)
org.apache.commons.io.output.DeferredFileOutputStream: java:103
org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader: java:1209)
java.lang.ClassLoader.loadClassInternal(ClassLoader: java:310)
org.apache.catalina.manager.HTMLManagerServlet.doPost(HTMLManagerServlet: java:157)
javax.servlet.http.HttpServlet.service(HttpServlet: java:803)
java.lang.ClassNotFoundException: org.apache.commons.io.output.DeferredFileOutputStream
org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader: java:1209)
java.lang.ClassLoader.loadClassInternal(ClassLoader: java:310)
org.apache.commons.io.output.DeferredFileOutputStream: java:103
org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader: java:1209)
java.lang.ClassLoader.loadClassInternal(ClassLoader: java:310)
org.apache.catalina.manager.HTMLManagerServlet.doPost(HTMLManagerServlet: java:157)
javax.servlet.http.HttpServlet.service(HttpServlet: java:803)
NOTE: The full stack trace of the root cause is available in the Apache Tomcat/5.5.26 logs.
Apache Tomcat/5.5.26
  
```

Thesis Overview

Legacy Mode

Runtime Exception
Analysis Framework

Exception
Localization
Or Injection

Resilient Catch
Taxonomy

Catch Synthesis

Catch Selection

Catch Injection

Resilience Mode

Runtime Exception Analysis

Applicative Goals

- analyze the exception behavior specification
- isolate the catch blocks needed to perform an action
- isolate the catch blocks which are not (or cannot be) used

Technical Goals

- detection of exception sources and their context
- capture of catch block execution and their interplay
- analyze the relationship between causes and treatments

The Spoon library is used to instrument source code

```

try{
    Framework.learn("try-start");
    //developer code
    Framework.learn("try-end");
}catch(RuntimeException re){
    Framework.learn("catch-start", re);
    //developer code
    Framework.learn("catch-end");
}
  
```

Results

Run org.apache.commons.lang Test Suite skipping the catch treatments: (2051 tests)

- 4 failures --> bad assert result
- 125 errors --> unexpected Exception
- 1 crash --> non-ending required Thread

Number of used catch blocks / thrown exceptions to boot

- Vuze: 24 catch – 413 exceptions
- Jabref: 5 catch – 50 exceptions
- Bluej: 0 catch – 1 exceptions

Ongoing Work

Exception injection of appropriate types at relevant place.

- select the place according to execution traces
- for a given place choose the type according to the static and dynamic context
- minimize artificiality / maximize learning

Evaluation

- How many “manually” fixed exception bugs from bug repositories can be automatically repaired by the framework?
- How much can the framework avoid crashing from fault injection?