Performance Analysis of Software System Versions (Performanzanalyse von Softwaresystemversionen)

David Georg Reichelt

Universität Leipzig

22. Juli 2015

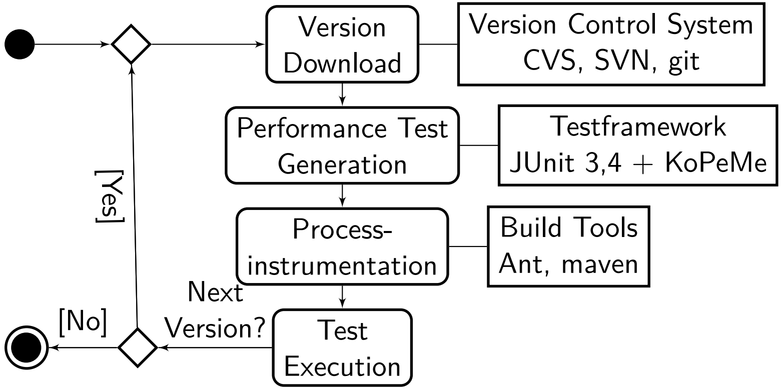
Basic Idea

- currently little empirical research on performance anti-patterns on code level
- basic assumption: performance of unit tests corresponds to performance of program
- approach: analyse performance of units tests of revisions of a program
  $\Rightarrow$ get performance problems
- goals
  - **derivation of performance problem classes on code-level**
  - quantification of occurence of performance problem classes

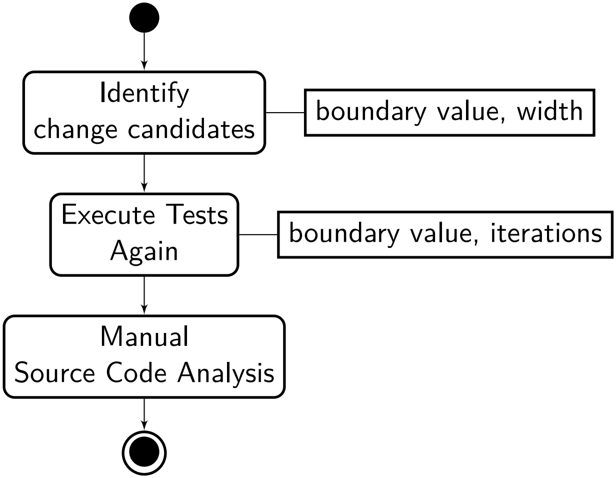
Steps

- steps
    - measurement of performance for all testcases in all revisions
    - identification of performance changes
    - identification of performance problems
- main problem: performance measurements are instable

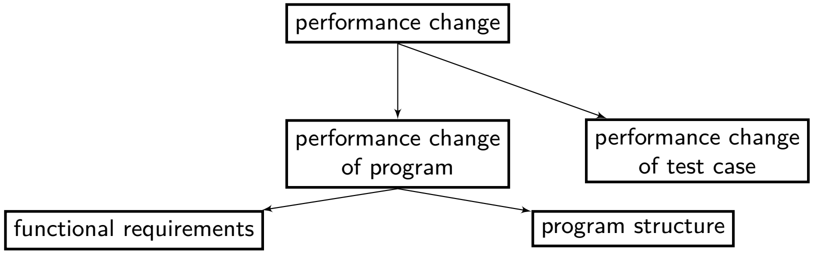
Performance Measurement



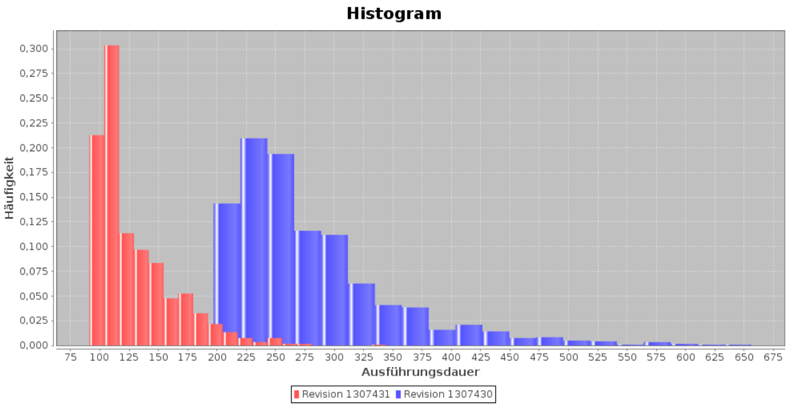
Identification of Performance Changes



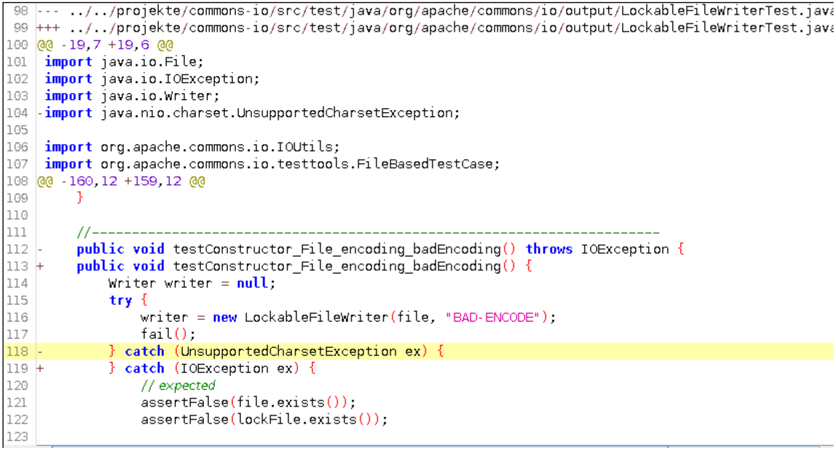
Type of Performance Changes


performance change
performance change of program
performance change of test case
functional requirements
program structure

Example Measurement



Example Diff



```
--- ../../projekte/commons-io/src/test/java/org/apache/commons/io/output/LockableFileWriterTest.jav
+++ ../../projekte/commons-io/src/test/java/org/apache/commons/io/output/LockableFileWriterTest.jav
@@ -19,7 +19,6 @@
 import java.io.File;
 import java.io.IOException;
 import java.io.Writer;
-import java.nio.charset.UnsupportedCharsetException;
 
 import org.apache.commons.io.IOUtils;
 import org.apache.commons.io.testtools.FileBasedTestCase;
@@ -160,12 +159,12 @@
     }
 
     //-----------------------------------------------------------------------
-    public void testConstructor_File_encoding_badEncoding() throws IOException {
+    public void testConstructor_File_encoding_badEncoding() {
         Writer writer = null;
         try {
             writer = new LockableFileWriter(file, "BAD-ENCODE");
             fail();
-        } catch (UnsupportedCharsetException ex) {
+        } catch (IOException ex) {
             // expected
             assertFalse(file.exists());
             assertFalse(lockFile.exists());
 
```

Shortcomings

- identification is manual process
  - error-prone
  - time-consuming

  ⇒ Root-Cause Isolation of Performance Regressions (Heger et al., 2013)
- measurement takes much time and is not reliable

  ⇒ use Kieker for test selection

Using Kieker for diff-analysis

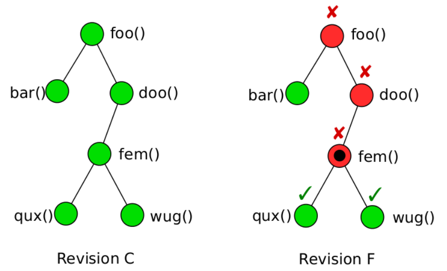


Abbildung : Call Tree Analysis from (Heger et al., 2013)

Using Kieker for diff-analysis

- Currently
  - javaassist-instrumentation of methods with kieker-measurement
  - reading the call tree
- Next Steps
  - measurement-dependent choice of next instrumented method(s)
  - use process in performance analysis of software system versions

Using Kieker for test selection

- Kieker $\Rightarrow$ Get call-tree of test-case
- VCS $\Rightarrow$ Get changed method
- only run test if call-tree of test contains changed method
  $\Rightarrow$ save time for other tests

Summary

- basic idea: examine development of performance of unit tests during software development
- goal: classification of typical performance problems
- usage of kieker
    - Root-Cause Isolation of Performance Regressions
    - detection of relevant test-cases