Plugging into Testovid

Ivan Pribela, Zoran Budimac

Content

- Automated assessment
- Testovid system
- Test anatomy
- Writing tests
- Conclusion

Content

- Automated assessment
- Testovid system
- Test anatomy
- Writing tests
- Conclusion

Automated assessment

- Assessment done by computers
- Assessment fazes
  - question / assignment selection
  - question / assessment deployment
  - answer / solution gathering
  - answer / solution grading
  - student / teacher notification
  - statistical analysis of results

Advantages (Large groups)

- Faster assessment
- Moving great amount of work load from teachers
- Leaving more time for more productive aspects of teaching

Advantages (Small groups)

- Enables self-assessment
- Teachers can easily note common problems and difficult topics
- Question selection can be better adjusted to individual needs
- Assessment results are immediate
- Same questions (assignments) can be repeated an much times as needed
- Questions can contain graphics, sound, animations, and other multimedia content
- Increase of assessment objectivity

Disadvantages

- Type of assessments supported
  - short multiple choice tests
  - fill-in-the-blanks tests with fixed correct answers
- Automated assessment is used more in natural sciences, and rarely in social sciences, where students write essays

Assessment of programming

- Programming assignments are perfect example for automated assessment
- Besides generic, special systems for assessing of computer programs were also developed
- There are two distinct types of computer program assessment:
  - correctness assessment
    - does the student program solve given problem?
  - optimality assessment
    - how efficient the student program is?

Assessing correctness

- Oldest correctness assessment method is
  - running student programs against various sets of input data
  - comparing output form student program with correct one
- Input data
  - created by teacher and given to the student
  - created by the student
  - hidden from the student
- Selecting input data
  - fixed
  - chosen randomly every time
- Output data
  - compared with fixed data corresponding to given input
  - compared with output from correct solution

Assessing optimality and style

- Early systems focused on execution speed and precision of numeric results
- Recently attention has shifted towards readability and code style
- Mostly based on absolute criteria
  - identifier length
  - line indentation
  - …
- Few attempts to use a program model

Content

- Automated assessment
- Testovid system
- Test anatomy
- Writing tests
- Conclusion

Testovid system

- Designed for assessment of programming
- Batch system
- Students can test solutions by themselves
- Teacher can test all assignments at once
- Generates execution reports
- Uses Apache Ant
- Designed to fit in Svetovid system

Apache Ant

- Apache Ant is a build process automation tool
- Similar to make, but written in Java
- Runs on Java virtual machine
- Most appropriate for Java projects
- Uses XML for description of build processes
- Open source
- Extensible

Implementation

- As a frame for domain-specific testers
- One domain-specific tester contains
  - Apache Ant file with testing modules
  - Configuration file with weights of every module
  - Any accompanying files
- Modular and extensible
- Previous tests can be reused

Content

- Automated assessment
- Testovid system
- Test anatomy
- Writing tests
- Conclusion

Configuration file

```
targets.all = test01, test02, test03,
    test04, test05, test06

test01.name = Compilation
test01.score= 1

test02.name = Up & Down methods
test02.score= 1

test03.name = Up – Upper restriction
test03.score= 2

test04.name = Down – Upper restriction
test04.score= 2

test05.name = Encapsulation
test05.score= 1

test06.name = Code style
test06.score= 1
```

- List of modules
- Name of every module
- Number of points for each module

- All marks are binary

Testing module implementation

```
<target name="test06">

 <checkstyle
  file="Assignment5.java"
  failureProperty="test06.fail"
  maxErrors="15"/>

 <condition
  property="test06.advice"
  value="There are more than 15
         style errors.">
  <equals
   arg1="${test06.fail} "
   arg2="true"/>
 </condition>

</target>
```

- One module is one Ant target
- Result of execution is string that contains message about an error
- Empty advice message means all is ok

Any accompanying files

- Input and output files
- Program model (correct solution)
- Configuration files
- Helper applications and scripts
- …

Content

- Automated assessment
- Testovid system
- Test anatomy
- Writing tests
- Conclusion

Writing tests

```
<target name="testX">

  <!-- Run intended tasks -->
  ...

  <!-- Set the advice -->
  <property
       name="testX.advice"
       value="This is the advice."/>

</target>
```

- One module is one target
- Write Ant target
- Run intended tasks
  - Compile student program
  - Run student program
  - Run correct solution
  - Check the outputs
  - Check student files
  - Check style
- Set advice or leave it blank

Types of tasks to use

- Built-in tasks
- 3rd party tasks
- Custom tasks
- Java application
- Native application
- Native script

Built-in tasks

```
<target name="test01">
  <trycatch>
    <try>

      <javac srcdir="." destdir="."
       source="1.5" target="1.5"/>

    </try>
    <catch>
      <property
       name="test01.advice"
       value="There are compilation
              errors."/>
    </catch>
    <finally/>
  </trycatch>
</target>
```

- Running Java compiler
  - javac
- Running Java application
  - java
- Executing Junit
  - junit
- Checking XML validity or well formedness
  - xmlvalidate
- Transforming XML
  - xslt

3rd party tasks

```
<target name="test02">

 <checkstyle
  file="Assignment5.java"
  failureProperty="test02.fail"
  maxErrors="15"/>

 <condition
  property="test02.advice"
  value="There are more than 15
         style errors.">
  <equals
   arg1="${test02.fail}"
   arg2="true"/>
 </condition>

</target>
```

- Checking Java source code style
  - checkstyle, jalopy
- Scanning for standard programming mistakes
  - pmd, xradar, hammurapi
- Invoking the ANTLR Translator generator
  - antlr

Custom task implementation

```
public class MyTask extends Task {

    private String advicePropertyName;

    public void setAdvice(
                String newValue) {
        advicePropertyName = newValue;
    }

    public void execute() {
        // Task implementation
    }

}
```

- Extend org.apache.tools.ant.Task
- For each attribute, write a setter method
- For each nested element, write an add method
- Write the execute method

Custom task implementation usage

```
<taskdef name="mytesttask"
         classname="MyTask"
         classpath="classes"/>
```

- Define the task
  - Task name
  - Task class
- Use the task under the defined name

```
<target name="test03">

  <mytesttask advice="test03.advice"/>

</target>
```

Custom Java application

```
<target name="test04">
  <trycatch>
    <try>

      <java classname="AnalyzeSolution"
            failonerror="yes"
            maxmemory="128m"/>

    </try>
    <catch>
      <property
       name="test04.advice"
       value="The produced solution is
              not correct."/>
    </catch>
    <finally/>
  </trycatch>
</target>
```

- Write standard Java application
- Run the application
  - Memory limit
  - Environment variables
  - Arguments

Any native application

```
<target name="test05">
  <trycatch>
    <try>

     <exec executable="diff">
      <arg line="out.txt correct.txt"/>
     </exec>

    </try>
    <catch>
      <property
       name="test05.advice"
       value="The program output is
              not correct."/>
    </catch>
    <finally/>
  </trycatch>
</target>
```

- Call native applications and commands
- Platform dependent
- No Java security manager

Or a native script

```
<target name="test06">
  <trycatch>
    <try>

     <exec executable="cmd"
           os="Windows">
      <arg line="/c test.bat"/>
     </exec>

    </try>
    <catch>
      <property
       name="test06.advice"
       value="The produced solution is
             not correct."/>
    </catch>
    <finally/>
  </trycatch>
</target>
```

- On Unix systems
  - directly
- On Windows or Cygwin
  - execute the shell
  - pass the batch file as argument using /c or -c switch
- More platform dependent
- No Java security manager

Content

- Automated assessment
- Testovid system
- Test anatomy
- Writing tests
- Conclusion

Conclusion

- Advantages
  - Platform independent
  - Flexible and powerful
  - Can be extended
  - Domain-specific testers and modules are reusable
- Disadvantages
  - Time must be invested to create testers
  - Knowledge of Apache Ant is needed

Questions?

Thank you for your attention