Integrated and Tool-Supported Teaching of Testing, Debugging, and Verification

Wolfgang Ahrendt, Richard Bubel, Reiner Hähnle

Chalmers University
Department of Computer Science and Engineering
Gothenburg
Sweden

November 16, 2009
Background

- authors:
  ‘Software Engineering with Formal Methods’ group at Chalmers

Ahrendt, Bubel, Hähnle (CTH)
Background

- authors:
  - ‘Software Engineering with Formal Methods’ group at Chalmers
- co-developed KeY tool for deductive Java source code verification
Background

- authors:
  ‘Software Engineering with Formal Methods’ group at Chalmers
- co-developed KeY tool for deductive Java source code verification
- we run Master level course:
  ‘Software Engineering using Formal Methods’
  using Spin and KeY
Background

- authors:
  ‘Software Engineering with Formal Methods’ group at Chalmers
- co-developed KeY tool for deductive Java source code verification
- we run Master level course:
  ‘Software Engineering using Formal Methods’
  using Spin and KeY
- we run 3rd year Bachelor level course:
  ‘Testing, Debugging, and Verification’
Background

- authors:
  ‘Software Engineering with Formal Methods’ group at Chalmers
- co-developed KeY tool for deductive Java source code verification
- we run Master level course:
  ‘Software Engineering using Formal Methods’
  using Spin and KeY
- we run 3rd year Bachelor level course:
  ‘Testing, Debugging, and Verification’

this talk: conception of the latter
Computing Education at Chalmers

Chalmers University of Technology

- Strong engineering tradition, most Swedish engineers from Chalmers
- Emphasis on traditional math courses: calculus, algebra, statistics
- Computing courses focus on programming
- on Bachelor level:
  No dedicated courses on theoretical computer science topics
Course Goals

Integration

FM as integrated aspect of quality code construction
Course Goals

Integration
FM as integrated aspect of quality code construction

Diversity
We present spectrum of quality ensuring activities:
error detection, error elimination, ensuring error freedom
Course Goals

Integration
FMs as integrated aspect of quality code construction

Diversity
We present spectrum of quality ensuring activities: error detection, error elimination, ensuring error freedom

Applicability
All methods in action with executable programs
Course Goals

Integration
  FMs as integrated aspect of quality code construction

Diversity
  We present spectrum of quality ensuring activities: error detection, error elimination, ensuring error freedom

Applicability
  All methods in action with executable programs

Formalisation ⇒ Tool Support
  Formalisation prerequisite for far-reaching analysis tools
Course Goals

Integration

FM as integrated aspect of quality code construction

Diversity

We present spectrum of quality ensuring activities: error detection, error elimination, ensuring error freedom

Applicability

All methods in action with executable programs

Formalisation ⇒ Tool Support

Formalisation prerequisite for far-reaching analysis tools

Tools are essential

Without tools potential of formalisation not fully realised
Exercise informal, but **precise** specification
Informal Specification

- Exercise informal, but *precise* specification
- Allegory of *specification as contract*
Exercise informal, but precise specification

Allegory of specification as contract

Raise awareness:
Exercise informal, but *precise* specification

Allegory of *specification as contract*

Raise awareness:

- if you write specs or not, *you always program towards contracts*
Informal Specification

- Exercise informal, but **precise** specification
- Allegory of **specification as contract**
- Raise awareness:
  - if you write specs or not, **you always program towards contracts**
- Example from **Java** programming:
Exercise informal, but precise specification

Allegory of specification as contract

Raise awareness:
- if you write specs or not, you always program towards contracts

Example from JAVA programming:
- All classes inherit contract from Object
Exercise informal, but precise specification

Allegory of specification as contract

Raise awareness:
  if you write specs or not, you always program towards contracts

Example from Java programming:
  All classes inherit contract from Object
  Object contract requires:
    hashcode() consistent with equals()
Exercise informal, but precise specification

Allegory of specification as contract

Raise awareness:
- if you write specs or not, you always program towards contracts

Example from Java programming:
- All classes inherit contract from Object
- Object contract requires:
  - `hashCode()` consistent with `equals()`
- Programmers/students typically break that contract
Exercise informal, but precise specification

Allegory of specification as contract

Raise awareness:
  - if you write specs or not, you always program towards contracts

Example from Java programming:
  - All classes inherit contract from Object
  - Object contract requires:
    - `hashCode()` consistent with `equals()`
  - Programmers/students typically break that contract
  - Consequence: collection classes malfunction
The TDV Course Structure

- Specification
  - Informal
The TDV Course Structure

Debugging

classic

Specification

Informal

classic testing

Testing
test inputs and test oracles based on informal specification
test inputs and test oracles based on informal specification

Contents

- White-/Blackbox testing
test inputs and test oracles based on informal specification

Contents

- White-/Blackbox testing
- Coverage criteria:
test inputs and test oracles based on informal specification

Contents

- White-/Blackbox testing
- Coverage criteria:
  - control flow graph coverage
test inputs and test oracles based on informal specification

Contents

- White-/Blackbox testing
- Coverage criteria:
  - control flow graph coverage
  - logic coverage
test inputs and test oracles based on informal specification

Contents

- White-/Blackbox testing
- Coverage criteria:
  - control flow graph coverage
  - logic coverage
  - input space partitioning
test inputs and test oracles based on informal specification

Contents

- White-/Blackbox testing
- Coverage criteria:
  - control flow graph coverage
  - logic coverage
  - input space partitioning
- Writing hand-crafted test cases
test inputs and test oracles based on informal specification

Contents

- White-/Blackbox testing
- Coverage criteria:
  - control flow graph coverage
  - logic coverage
  - input space partitioning
- Writing hand-crafted test cases

Tool: JUnit
**Debugging**

Disregarded by most software engineering lectures, in contrast to development time actually spent on debugging.


**Classic Debugging Techniques**

- Logging of events
- Controlled execution: Step into/over, breakpoints
- Inspection: variable values, heap inspection

**Tools:** log4j, eclipse debugger

**Delta Debugging**

- Automatic retrieval of minimal input triggering the bug

**Tool:** DDinput
The TDV Course Structure

- Debugging
  - classic

Specification
  - Informal

- Testing
  - classic testing
Students learn

- Formalisation of real-world problems,
- Basics of first-order logic and
- Java Modelling Language (JML) as specification language

**Tools:** jml (syntax and type checker, Common JML Tools)

Formal specification prerequisite for automation of

- Test generation
- Symbolic debugging
- Formal verification
The TDV Course Structure

Debugging

classic

Specification

Informal

Formal

classic testing

Testing

Ahrendt, Bubel, Hähnle (CTH) Testing, Debugging, and Verification November 16, 2009 12 / 19
The TDV Course Structure

- Debugging
  - classic

- Specification
  - Informal
  - Formal
    - model-based
    - ATCG

- Testing
  - classic testing
The TDV Course Structure

- Debugging
  - classic
  - visual

- Specification
  - Informal
  - Formal

- Formal Verification
  - Symbolic Execution

- Testing
  - classic testing
  - model-based ATCG
  - code-based ATCG

Ahrendt, Bubel, Hähnle (CTH) Testing, Debugging, and Verification November 16, 2009 12 / 19
Automatic Test Generation

Blackbox Testing: Model-based testing
Based on formal specification

- Coverage criteria incl. specifications
- Derivation of test scenarios/cases by disjunction analysis
- Deriving test cases from JML specifications

Tool: jmlunit (Common JML Tools)
Automatic Test Generation

Blackbox Testing: Model-based testing
Based on formal specification

- Coverage criteria incl. specifications
- Derivation of test scenarios/cases by disjunction analysis
- Deriving test cases from JML specifications

**Tool:** jmlunit (**Common JML Tools**)  

White-box testing
Test cases derived from

- Formal specification and
- Source code

Introducing *symbolic execution* as basic technology.  

**Tool:** KeY-VBT
Symbolic Debugging

Based on symbolic execution

- Covers all possible execution paths
- No initialisation necessary
- Efficient omniscient debugging
Symbolic Debugging

Before removal:

- Symbolic heap inspection
- Specification constrains valid heap configuration

After removal:

- Based on symbolic execution
- Covers all possible execution paths
- No initialisation necessary
- Efficient omniscient debugging
- Symbolic heap inspection
Symbolic Debugging

Intended configuration:

```
list_0:List → next → list_1:List → next → list_2:List
```

Unintended configuration:

```
list_0:List → next
```

Based on symbolic execution
- Covers all possible execution paths
- No initialisation necessary
- Efficient omniscient debugging
- Symbolic heap inspection
  - Specification constrains valid heap configuration
Verification

Most formal approach taught in TDV.
Verification

**Most formal** approach taught in TDV.
Calculus and tool developed specifically for that course.

**Hoare calculus with explicit substitutions**

Hoare calculus variant based on *symbolic execution*
- forward reasoning
- elimination of most non-deterministic rules
- first-order reasoning as black-box
Verification

Most formal approach taught in TDV.
Calculus and tool developed specifically for that course.

Hoare calculus with explicit substitutions
Hoare calculus variant based on symbolic execution
- forward reasoning
- elimination of most non-deterministic rules
- first-order reasoning as black-box

Tool: KeY-Hoare
- Interactive and automatic verification system (based on KeY)
- Powerful first-order/arithmetic proving capabilities
- Supports partial, total, and execution time aware correctness
Experiences and Discussion

Course given first time: Summer 2007
Course Name: Program verification
Participants: 15 students

Renamed to Testing, Debugging and Verification in 2008
Participants: 80 students

Evaluation
Course has been rated high by participating students.
90% of students completed course (high rate for non-compulsory course)

Limitations
Compromise between available time, topic and depth
Missing:
- Software certification and code reviews
- Integration into software development process (planned!)

Course adapted by U. of Innsbruck, TU of Madrid and U. of Freiburg
Experiences and Discussion

Course given first time: Summer 2007
Course Name: Program verification
Participants: 15 students

Renamed to Testing, Debugging and Verification in 2008
Participants: 80 students
Experiences and Discussion

Course given first time: Summer 2007
Course Name: Program verification
Participants: 15 students

Renamed to Testing, Debugging and Verification in 2008
Participants: 80 students

Evaluation
Course has been rated high by participating students.
90% of students completed course (high rate for non-compulsory course)
Experiences and Discussion

Course given first time: Summer 2007
Course Name: Program verification
Participants: 15 students

Renamed to Testing, Debugging and Verification in 2008
Participants: 80 students

Evaluation
Course has been rated high by participating students.
90% of students completed course (high rate for non-compulsory course)

Limitations

- Compromise between available time, topic and depth
- Missing:
  - Software certification and code reviews
  - Integration into software development process (planned!)
Experiences and Discussion

Course given first time: Summer 2007
Course Name: Program verification
Participants: 15 students

Renamed to Testing, Debugging and Verification in 2008
Participants: 80 students

Evaluation
Course has been rated high by participating students.
90% of students completed course (high rate for non-compulsory course)

Limitations
- Compromise between available time, topic and depth
- Missing:
  - Software certification and code reviews
  - Integration into software development process (planned!)

Course adapted by U. of Innsbruck, TU of Madrid and U. of Freiburg
Can a Bachelor level course be research driven?

Our research objective is precisely increased accessibility of FMs. Students profit from this objective, and we profit from increased pressure on usability.
Can a Bachelor level course be research driven?
Can a Bachelor level course be research driven?

- Our research objective is precisely increased accessibility of FMs
Can a Bachelor level course be research driven?

- our research objective is precisely increased accessibility of FMs
- students profit from this objective
Can a Bachelor level course be research driven?

- our research objective is precisely increased accessibility of FMs
- students profit from this objective
- we profit from increased pressure on usability
Adaption

in spite of the close connection to our own research: 

course can be run in any other context

- All tools freely available, mostly open source
- We actively support adaptation of course (units)
- Course parts adapted at:
  - Technical University of Madrid
  - University of Innsbruck
  - University of Freiburg
- Feedback from adaptations improved our course 
  (e.g. worst-case execution time in KeY-Hoare suggested by Joanna Chimiak-Opoka, Innsbruck)
### Overview: Tool-Based Teaching

<table>
<thead>
<tr>
<th>Teaching Unit</th>
<th>Content</th>
<th>Formal</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing</td>
<td>Systematic testing, specification, assertions, black/white box, path/code coverage</td>
<td>no</td>
<td>JUNIT</td>
</tr>
<tr>
<td>Debugging</td>
<td>Bug tracking, execution control, failure input minimisation, logging, slicing</td>
<td>no</td>
<td>DDinput, Eclipse, log4j</td>
</tr>
<tr>
<td>Formal Specification</td>
<td>Design-by-contract, formalisation, first-order logic, JML</td>
<td>yes</td>
<td>jml</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(type checker)</td>
</tr>
<tr>
<td>Automated Test Case</td>
<td>Model-based TC generation, Symbolic execution, Code-based TC generation</td>
<td>yes</td>
<td>jmlunit, Key VSD, Key VBT</td>
</tr>
<tr>
<td>Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formal Verification</td>
<td>Hoare triple, weakest precondition, formal verification, loop invariant</td>
<td>yes</td>
<td>KeY-Hoare</td>
</tr>
</tbody>
</table>

All tools freely available software and most open source.