From Design by Contract to Static Analysis of Java Programs: A Teaching Approach

FM-Ed 2006

Dr. Christelle Scharff, Pace University, NY, USA

Sokharith Sok, Pace University, NY, USA & Institute of Technology of Cambodia

Correctness

- Programmers should be able to answer two fundamental questions about their program:

  – What is the meaning of the program they developed?

  – Can it be proved that it has the correct meaning?

  Programmers are not required to write the proofs but should be provided with tools that would help them guarantee that their programs meet given specifications.

  [Lamport, 1977]

Approach (1)

- A Design-by-Contract-First approach
- Covering the same problems redundantly
  - Informally
  - Using Design by Contract
  - Using Static Analysis for software verification

Approach (2)

- **Step 1:** Determine normal courses, alternative courses, exceptions, error cases and boundary conditions

```
/*
* Factorial computes factorial n
* @param n int
* @return int - the factorial of n
*/
public int factorial(int n) {
    if (n == 0) {
      return 1;
    }
  else {
     return n * factorial(n - 1);
   }
}

/*
* @param n int - needs to be <= 13 and > 0
* @return int - the factorial of n
* @exception IllegalArgumentException on bad inputs
*/
public int factorial(int n) {
    if (n <= 13 && n > 0) {
        return n * factorial(n - 1);
    } else if (n == 0) {
        return 1;
    }
    throw new IllegalArgumentException("argument pb");
}
```

Approach (3)

- **Step 2:** Determine the pre-conditions, post-conditions and loop (and class) invariants
- **Step 3:** Design by Contract approach [Meyer, 1992]

  – Assertions written in Java 1.5 in the Eclipse integrated development environment

  – Pre-conditions - *exceptions*

  – Post-conditions - *assert*

  – No class invariants

  – Agile Java

  P*: if we start with x equals to y and execute $x = x + 1$ followed by $y = y + 1$, then x is still equal to y.*

```
// Pre-condition
if (x != y){
    throw new illegalArgumentException("x must be equal to y");
}
x = x + 1;
y = y + 1;
// Post-condition
assert (x == y): "After execution of the program X = " + x +
" must be equal to Y = " + y + ".";
```

Approach (4)

- **Step 4:** Introduction to Static Analysis
  - E.g. Unnecessary else, unused variables and methods
  - Advantages (e.g. execution time)

- **Step 5:** Static Analysis for Software Verification
  - Hoare Logic [Hoare, 1969]
  - Annotation of the code with pre-conditions, post-conditions, invariants written in particular logic
  - Propagation of post-conditions – Done on paper!
  - Check of the assertions using ICS (I Can Solve / Integrated Canonizer and Solver), a little engine of proofs developed at SRI

  [http://www.icansolve.com]

[x = y] (Pre-condition)

```
[x + 1 = y + 1]  (Generated pre-condition)
x = x + 1;
[x = y + 1]      (Generated assertion)
y = y + 1;
```

[x = y] (Post-condition)

```
ics>  sat x=y & ~x+1 =y+1.
:unsat
```

Implementation of the Approach

- *Programming Languages* course of 15 junior students in fall 2005
  - 2 weeks
- Assessment: Homework (toy examples) + Questionnaire
  - As a developer, what do you think are the difficulties with annotating programs with assertions that are checked at run-time?
  - As a developer, what do you think are the difficulties of formal static analysis of programs?
  - What would you be more willing to use as a developer: design by contract or formal static analysis? Why?
  - In formal static analysis of programs, what do you think are the phases that are the responsibility of the developer and what should be automated?

Students Difficulties

- How to come up with the right pre-conditions, post-conditions and invariants?
- Design by Contract easier because integrated and closer to the code
- Static analysis phase
  - Lack of automation
    - Need of integrated tools
  - Lack of fluency in logic
  - Scalability concerns
  - Scope of the tool used to check assertions
    - What the tool can prove, what it cannot prove, what language it is based on?

Conclusion

- Main concerns:
  - Difficulties with logic
  - Tools for formal software verification
- "Given the right tools, the use of formal methods could become widespread and transform software engineering". C. Jones, P. O'Hearn and J. Woodcock. *Verified Software: A Grand Challenge*. IEEE Computer Magazine article, April 2006.