Testing Meets Static and Runtime Verification

Jesús Mauricio Chimento, Wolfgang Ahrendt, Gerardo Schneider

FormaliSE’18

2 June 2018
Software Development

Testing

Formal Verification
Software Development

Testing

- Test Driven Development
- Behaviour Driven Development
Test Driven Development

- Write (unit) test cases which initially fail
- Write code making the tests pass
- *Refactor* the code
Example

- Write (unit) test cases which initially fail

```java
/**
 * Deletes entry at <tt>key</tt> from the hashtable.
 *
 * @param key of the removed object
 * @return removed object
 */

public Object delete (int key) { }
```
Example

- Write (unit) test cases which initially fail

```java
/**
 * Deletes entry at <tt>key</tt> from the hashtable.
 *
 * @param key of the removed object
 * @return removed object
 */

public Object delete (int key) { }
```

```java
@Test
public void test_delete_1(){
    hash.add(new Integer(42),0);
    hash.add(new Integer(3),1);
    HashTable aux = new HashTable(2);
    aux.add(new Integer(3),1);
    Object res = hash.delete(0);
    assertEquals(res,new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```
Example

- Write code making the tests pass

```java
/**  
 * Deletes entry at <tt>key</tt> from the hashtable.
 *  
 * @param key of the removed object  
 * @return removed object  
 */

public Object delete (int key) {
    if (key >= 0) {  
        if (h[key] == null)
            return null;
        else {
            Object ret = h[key] ;
            h[key] = null ;
            size = size - 1;
            return ret;
        }
    } else { return null; }
}

@Test
public void test_delete_1(){  
    hash.add(new Integer(42),0);
    hash.add(new Integer(3),1);
    HashTable aux = new HashTable(2);
    aux.add(new Integer(3),1);
    Object res = hash.delete(0);
    assertEquals(res,new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```
Behaviour Driven Development

- **Red - Green - Refactor**
- Scenarios instead of unit tests
Behaviour Driven Development

- Property: deposit available only when user is logged

```java
/**
 * Deposits money in user's account.
 * @param money amount of money to deposit
 */
public void deposit(int money){
}
```
Behavour Driven Development

- Property: deposit available only when user is logged

```java
/**
 * Deposits money in user's account.
 * *
 * @param money amount of money to deposit
 */
public void deposit(int money){
    if (u != null)
        u.getAccount().deposit(money);
}
```
**Behaviour Driven Development**

- Property: deposit available only when user is logged

```java
/**
 * Deposits money in user's account.
 * @param money amount of money to deposit
 */
public void deposit(int money){
    if (u != null)
        u.getAccount().deposit(money);
}
```
Behaviour Driven Development

- Property: deposit available only when user is logged

```java
/**
 * Deposits money in user's account.
 * @param money amount of money to deposit
 */
public void deposit(int money){
    if (u != null)
        u.getAccount().deposit(money);
}
```

```
Logout

u != null \ deposit \ u != null

u == null \ login \ u != null

Login

u != null \ logout \ u == null
```

```java
logout

Logout

u != null \ deposit \ u != null

u == null \ login \ u != null

Login

u != null \ logout \ u == null
```

```java
logout

Logout

u != null \ deposit \ u != null

u == null \ login \ u != null

Login

u != null \ logout \ u == null
```

```java
/**
 * Deposits money in user's account.
 * @param money amount of money to deposit
 */
public void deposit(int money){
    if (u != null)
        u.getAccount().deposit(money);
}
```
Software Development

Formal Verification

Deductive Verification

Runtime Verification
Deductive Verification

- Properties written as logical formulae

\{ P \} \text{foo()} \{ Q \}

- Formulae are verified by deduction in a calculus

\[
\Gamma, \sigma(b) \vdash \sigma <s_1 \omega > \phi \\
\Gamma, \sigma(\neg b) \vdash \sigma <s_2 \omega > \phi \\
\Gamma \vdash \sigma <\text{if } b \ s_1 \text{ else } s_2 \omega > \phi
\]
Runtime Verification

• Monitoring of program executions

Diagram:
- Specification \( S \)
- Program
- Runtime Verifier
- Monitor
- Program (weaved)
Test Focus Development

Test Driven Development

Behaviour Driven Development

Deductive Verification

Runtime Verification
Test Focus Development

- Test Driven Development
- Behaviour Driven Development
- Deductive Verification
- Runtime Verification
Test Focus Development

- Test Driven Development
- Model-Based Testing
- Model Definition
- Deductive Verification
- Runtime Verification
Test Focus Development

Methods Signature Definition → Test Driven Development → Model Definition → Model-Based Testing

Data + Control

Deductive Verification + Runtime Verification

Model
Testing Meets Deductive and Runtime Verification

- Contracts Definition
- Test Driven Development
- Deductive Verification
- Model Definition
- Model-Based Testing
- Overall Implementation
- Runtime Verification
- Methods Signature Definition
- Test Cases
- data
- control
Example

- Define the methods signature

```java
/**
 * Deletes entry at `<tt>key</tt>` from the hashtable.
 * @param key of the removed object
 * @return removed object
 */
public Object delete (int key) { }
```
Define contracts for the methods

```java
/*@ public normalBehaviour
  @ requires key >= 0 ;
  @ requires h[hash(key)] != null ;
  @ requires size > 0 ;
  @ ensures \result == \old(h[hash(key)]) ;
  @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
  @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
  @ assignable size,h[*] ;
  @ also
      ..
  @*/

public Object delete (int key) { }
```
Example

- Apply TDD (at least one test per contract)

```java
/**
* public normal behaviour
* @ requires key >= 0 ;
* @ requires h[hash(key)] != null ;
* @ requires size > 0 ;
* @ ensures \result == \old(h[hash(key)]) ;
* @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
* @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
* @ assignable size,h[*] ;
* @ also
* ......
* @*/
public Object delete (int key) { }

@Test
public void test_delete_1(){
    hash.add(new Integer(42),0);
    hash.add(new Integer(3),1);
    HashTable aux = new HashTable(3) ;
    aux.add(new Integer(3),1);
    Object res = hash.delete(0);
    assertEquals(res,new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```
Example

- Apply TDD (at least one test per contract)

```java
/*@ public normal_behavior
@ requires key >= 0 ;
@ requires h[hash(key)] != null ;
@ requires size > 0 ;
@ ensures result == \old(h[hash(key)]) ;
@ ensures h[hash(key)] == null && size == \old(size) - 1 ;
@ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
@ assignable size,h[*] ;
@ also
....
@*/
public Object delete (int key) {
    if (key >= 0) {
        if (h[key] == null)
            return null;
        else {
            Object ret = h[key];
            h[key] = null;
            size = size - 1;
            return ret;
        }
    } else { return null; }
}
```

```java
@Test
public void test_delete_1(){
    hash.add(new Integer(42),0);
    hash.add(new Integer(3),1);
    HashTable aux = new HashTable(3) ;
    aux.add(new Integer(3),1);
    Object res = hash.delete(0);
    assertEquals(res,new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```
Example

- Deductive verify the implementation

```java
/*@ public normal_behaviour
 @ requires key >= 0 ;
 @ requires h[hash(key)] != null ;
 @ requires size > 0 ;
 @ ensures result == \old(h[hash(key)]) ;
 @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
 @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
 @ assignable size,h[*] ;
 @ also
 ......
 @*/
 public Object delete (int key) {
     if (key >= 0) {
         if (h[key] == null)
             return null;
         else {
             Object ret = h[key];
             h[key] = null;
             size = size - 1;
             return ret;
         }
     } else { return null; }
 }
```
• Proof-based test case generation

```java
/*@ public normal_behaviour
   @ requires key >= 0 ;
   @ requires h[hash(key)] != null ;
   @ requires size > 0 ;
   @ ensures \result == \old(h[hash(key)]) ;
   @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
   @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
   @ assignable size,h[*] ;
   @ also
       ..... 
   */
public Object delete (int key) {
    if (key >= 0) {
        if (h[key] == null)
            return null;
        else {
            Object ret = h[key];
            h[key] = null;
            size = size - 1;
            return ret;
        }
    } else { return null; }
}
```
Example

- Proof-based test case generation

```java
/*@ public normal Behaviour
@ requires key >= 0 ;
@ requires h[hash(key)] != null ;
@ requires size > 0 ;
@ ensures \result == \old(h[hash(key)]) ;
@ ensures h[hash(key)] == null && size == \old(size) – 1 ;
@ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
@ assignable size,h[*] ;
@ also
.....
*/
public Object delete (int key) {
    if (key >= 0) {
        if (h[key] == null)
            return null;
        else {
            Object ret = h[key];
            h[key] = null;
            size = size - 1;
            return ret;
        }
    } else { return null; }
}
```

*KeyTestGen* generates a (failing) test case which throws an index out of bound exception.
Example

- Proof-based test case generation

```java
/*@ public normalBehaviour
@ requires key >= 0 ;
@ requires h[hash(key)] != null ;
@ requires size > 0 ;
@ ensures \result == \old(h[hash(key)]) ;
@ ensures h[hash(key)] == null && size == \old(size) – 1 ;
@ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
@ assignable size,h[*] ;
@ also
.....
@*/
public Object delete (int key) {
    if (key >= 0) {
        int i = hash(key);
        if (h[i] == null)
            return null;
        else {
            Object ret = h[i];
            h[i] = null;
            size = size - 1;
            return ret;
        }
    } else { return null; }
}
```

KeyTestGen generates a (failing) test case which throws an index out of bound exception.
Example

Proof-based test case generation

```java
/*@ public normal_behaviour
   @ requires key >= 0 ;
   @ requires h[hash(key)] != null ;
   @ requires size > 0 ;
   @ ensures \result == \old(h[hash(key)]) ;
   @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
   @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
   @ assignable size,h[*] ;
   @ also
   .....
   */

public Object delete (int key) {
   if (key >= 0) {
      int i = hash(key);
      if (h[i] == null)
         return null;
      else {
         Object ret = h[i];
         h[i] = null;
         size = size - 1;
         return ret;
      }
   } else { return null; }
}

@Test
public void test_delete_1(){
   hash.add(new Integer(42),0);
   hash.add(new Integer(3),1);
   HashTable aux = new HashTable(3) ;
   aux.add(new Integer(3),1);
   Object res = hash.delete(0);
   assertEquals(res,new Integer(42));
   assertNull(hash.get(0));
   assertTrue(hash.size == 1);
   assertArrayEquals(aux.h, hash.h);
}
```
Testing Meets Deductive and Runtime Verification

- Contracts Definition
- Test Driven Development
  - Test Case Generation (symbolic execution)
  - (partial) Proofs
  - Deductive Verification
    - Contracts Verification
      - Test Cases
- Model Definition
- Model-Based Testing
- Overall Implementation
- Runtime Verification

Data

Control
Example

- Define the model for your (control) property

```
u != null \ deposit \ u != null
```

```
u == null \ login \ u != null
```

```
u != null \ logout \ u == null
```

```
u != null \ withdraw \ u != null
```
Example

- Use MBT to develop the methods

```java
public void deposit(int money) {
    if (u != null)
        u.getAccount().deposit(money);
}
```

/**
 * Deposits money in user's account.
 *
 * @param money amount of money to deposit
 */

u != null \ deposit \ u != null

u == null \ login \ u != null

u != null \ logout \ u == null

u != null \ withdraw \ u != null
Example

- Finish the overall implementation

  (i.e. implement method *main*)

```java
switch (inputLine) {
    case "deposit":
        System.out.print("Enter amount to deposit: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
    case "withdraw":
        System.out.print("Enter amount to withdraw: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
```
Example

- Finish the overall implementation
  (i.e. implement method \textit{main})

```java
switch (inputLine) {
    case "deposit":
        System.out.print("Enter amount to deposit: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
    case "withdraw":
        System.out.print("Enter amount to withdraw: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
}
```
• Use runtime verification to validate the model

```
u != null \ deposit \ u != null

u == null \ login \ u != null

u != null \ logout \ u == null

u != null \ withdraw \ u != null
```
Example

- Use runtime verification to validate the model
Example

- Execute the monitor against MBT traces

```java
switch (inputLine) {
    case "deposit":
        System.out.print("Enter amount to deposit: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
    case "withdraw":
        System.out.print("Enter amount to withdraw: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
```
Example

- Execute the monitor against MBT traces

```java
switch (inputLine) {
    case "deposit":
        System.out.print("Enter amount to deposit: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.deposit(aux);
        break;
    case "withdraw":
        System.out.print("Enter amount to withdraw: ");
        amount = in.next();
        aux = Integer.parseInt(amount);
        f.withdraw(aux);
        break;
}```
• Extending the monitor

```java
public void deposit(int money)
{
    if (u != null)
        u.getAccount().deposit(money);
}
```

```
// safety data integrity
u == null \ login \ u != null
```

```
logout
```

```
BAD
```

```
public void deposit(int money){
    if (u != null)
        u.getAccount().deposit(money);
}
Testing Meets Deductive and Runtime Verification

Contracts Definition → Test Driven Development

Methods Signature Definition → Deductive Verification

Contracts Verification → (partial) Proofs

Test Case Generation (symbolic execution)

Model Definition → Runtime Verification

Monitor Generation → Monitor Spec.

Monitor Extension → Model Translation

Model Translation → Overall Implementation
Usage Remarks

Contracts Definition → Test Driven Development

Methods Signature Definition

Deductive Verification

Contracts Verification

(partial) Proofs

Test Case Generation (symbolic execution)

Test Cases

Model Definition

Runtime Verification

Monitored Execution

Monitor Generation

Monitor Spec.

Monitor Extension

Model Translation

Model-Based Testing

Overall Implementation
Conclusions

- Test focus development technique enhanced with formal verification
- (Static) deductive verification enhances TDD when dealing with data aspects
- Runtime Verification enhances MBT when dealing with control aspects
- Compositional usage of the different parts of the proposed technique