Enhancing Software Development with IDE-managed Codebases

Kıvanç Muşlu, University of Washington

© Kıvanç Muşlu, University of Washington, 2013
A copy of the developer’s codebase can enhance software development by adding support for
1. continuous analysis (e.g., testing) execution
2. systematic exploration of the history
3. better maintenance of multiple codebases
Outline

• Motivating example
• Copy codebase and replication framework
  – IDE-integrated continuous analyses
  – Fine-grained development history
  – Layers: better support for multiple codebases
• Contributions
Motivating Example

```java
public class Math {
    static double divide(double n1, double n2) throws DivideByZeroError {
        if (n2 == 0) {
            throw new DivideByZeroError();
        }
        return n1 / n2;
    }

    static double add(double n1, double n2) {
        throw new RuntimeException("Not supported");
    }
}

public class MathTest {
    @Test public void divideByZero() {
        Assert.assertEquals(Double.NaN, Math.divide(1.0, 0.0));
    }

    @Test public void add() {
        Assert.assertEquals(2.0, Math.add(1.0, 1.0));
    }
}
```
Implementing the Feature

```java
public class Math {
    static double divide(double n1, double n2) throws DivideByZeroError {
        if (n2 == 0)
            throw new DivideByZeroError();
        return n1 / n2;
    }

    static double add(double n1, double n2) {
        return n1 + n2;
    }
}
```

```java
public class MathTest {
    @Test public void divideByZero() {
        Assert.assertEquals(Double.NaN, Math.divide(1.0, 0.0));
    }

    @Test public void add() {
        Assert.assertEquals(2.0, Math.add(1.0, 1.0));
    }
}
```

Branch: feature
Implementing the Feature

- Developer invokes the analysis manually, waits for the results
  - Interrupts development process

```java
public class Math {
    static double divide(double n1, double n2) throws DivideByZeroError {
        if (n2 == 0)
            throw new DivideByZeroError();
        return n1 / n2;
    }

    static double add(double n1, double n2) {
        return n1 + n2;
    }
}
```

Branch: feature
Problem 1: It IS difficult to implement continuous analyses

- Deal with concurrent developer edits
  - Incrementalization: eclipse compiler
  - Running on the current file only: FindBugs [HovemeyerP 2004]
  - Changing IDE API: Continuous Testing [SaffE 2005], Quick Fix Scout [MusluBHEN 2012]

- Maintains a copy codebase
  Quick Fix Scout [MusluBHEN 2012], Crystal [BrunHEN 2011], Beacon*

Solution: Copy codebase w/o interference

- Wrap non-continuous analyses into continuous ones

* Implemented at Microsoft Research, Summer 2011
Replication framework: Incremental maintenance of copy codebase
Replication framework: Incremental maintenance of copy codebase
Replication framework: Incremental maintenance of copy codebase
Replication framework: Incremental maintenance of copy codebase
Research Questions [MusluBEN 2013, in submission]

• What is the overhead for a copy codebase?
  – Is it feasible? Yes, if incremental: \( \sim 6 \) ms / edit
    [Avg touch typist, 40 WPM = \( \sim 333 \) ms / edit => \( \sim 1.8\% \) overhead]

• What is the effort to wrap a non-continuous analysis into a continuous one?
  – Is it usable? Yes, \( \sim 500 \) LoC & 1-2 weeks / analysis
    [Quick Fix Scout: 7.5 KLoC, 10 months]
Outline

• Motivating example
• Copy codebase and replication framework
  – IDE-integrated continuous analyses
  – Fine-grained development history
    (Required for systematic exploration)
  – Layers: better support for multiple codebases
• Contributions
Motivating Example

```java
public class Math {
    static double divide(double n1, double n2) throws DivideByZeroError {
        if (n2 == 0)
            throw new DivideByZeroError();
        return n1 / n2;
    }

    static double add(double n1, double n2) {
        throw new RuntimeException("Not supported");
    }
}

public class MathTest {
    @Test public void divideByZero() {
        Assert.assertEquals(Double.NaN, Math.divide(1.0, 0.0));
    }

    @Test public void add() {
        Assert.assertEquals(2.0, Math.add(1.0, 1.0));
    }
}
```

Branch: main
Fixing the Bug

• Implementation and test are correct internally
  – However, they use different specs
• History might identify the correct spec, if present

Branch: main
Problem 2: Development history MIGHT not be available

- Harder to find regression bugs
- Harder to understand how software evolved
Problem 2: Development history MIGHT get lost

Reason: History management is mostly manual

Solution: Store a fine-grained history automatically

Storyteller VCS [Mahoney 2012], Mylyn [KerstenM 2005], IDE++ [Gu 2012]
Fine-grained History Framework

- Replication framework detects developer edits
- Create a repository in copy codebase, commit changes periodically
Research Questions

• Overhead for creating a fine-grained history?
  – We expect the amortized cost will be acceptable

• Application areas for a fine-grained history?
  – Systematic exploration of history: Dependency-Aware, selective, Tree-based Undo Model [refer to paper]
  – Improving accuracy of research on history & actions
    • Mining software repositories [NagappanZZHM 2010]
    • Improving IDE recommendations based on historical actions [BruchMM 2009]
Outline

• Motivating example
• Copy codebase and replication framework
  – IDE-integrated continuous analyses
  – Fine-grained development history
  – Layers: better support for multiple codebases
• Contributions
Implementing the Fix

public class Math
{
    static double divide(double n1, double n2) throws DivideByZeroError {
        if (n2 == 0)
            throw new DivideByZeroError();
        return n1 / n2;
    }

    static double add(double n1, double n2) {
        throw new RuntimeException("Not supported");
    }
}

public class MathTest
{
    @Test(expected = DivideByZeroError.class)
    public void divideByZero() {
        Math.divide(1.0, 0.0);
    }

    @Test
    public void add() {
        Assert.assertEquals(2.0, Math.add(1.0, 1.0));
    }
}

Branch: main
Problem 3: It IS difficult to maintain multiple development codebases

- Developer manually propagates changes
- When branches diverge, conflicts might occur
Problem 3: It IS difficult to maintain multiple development codebases

Reason: VCS branches do not encode the relation between development codebases

Solution for multiple developers and products: SPL [PohIBL 2005]

Solution: Scale down Software Product Lines
- Single developer
- Small tasks
- Faster feedback

Layers: parent-child relation between development codebases
Layers

Branch: main

@Test(expected = DivideByZeroError.class) public void divideByZero() {
    Math.divide(1.0, 0.0);
}

public class Math {
    static double divide(double n1, double n2) throws 
    if (n2 == 0)
        throw new DivideByZeroError();
    return n1 / n2;
}

static double add(double n1, double n2) { 
    return n1 + n2;
}

@Test(expected = DivideByZeroError.class) public void divideByZero() {
    Math.divide(1.0, 0.0);
}

@Test public void add() {
    Assert.assertEquals(2.0, Math.add(1.0, 1.0));
}

Branch: feature
Map Layer operations to VCS operations, execute on copy codebase
Research Questions

• What is the overhead for Layer operations?
  ✓ Simple operations (create, split, merge)
  ? Complex operations (run same analysis on multiple layers)

• Evaluate usefulness with case studies
  – Does number of conflicts reduce?
  – Can developers detect conflict earlier?
  – Does auto-propagation make development faster?
Contributions

• Replication framework design
  – Analyses run on copy codebase w/o interference
  – Continuous analyses implementation easier

• Use cases for Replication framework
  – Maintaining fine-grained history
    • Improves research based on historical actions & code
    • Systematic exploration of the history
  – Layers: Better multiple code maintenance
References


References


