Enhancing Software Development with IDE-managed Codebases

Kıvanç Muşlu, University of Washington

© Kıvanç Muşlu, University of Washington, 2013

Thesis Statement

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

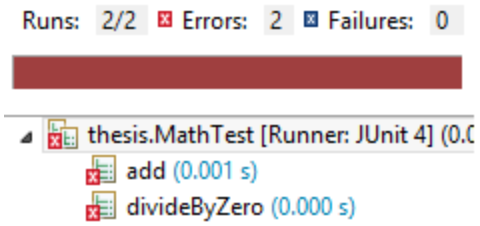
Motivating Example

```
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");
    }
}
```

2. Fix bug

1. Implement feature

```
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));
    }
}
```

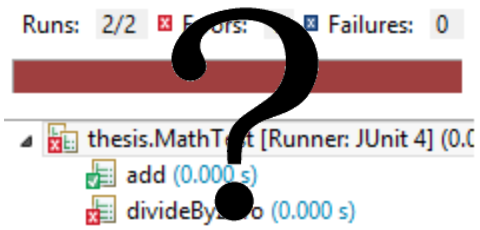


Branch: main

Implementing the Feature

```
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;
    }
}
```

```
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));
    }
}
```

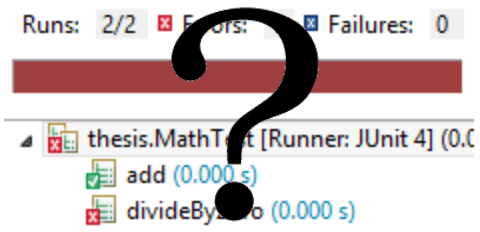


Branch: feature

Implementing the Feature

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

```
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*

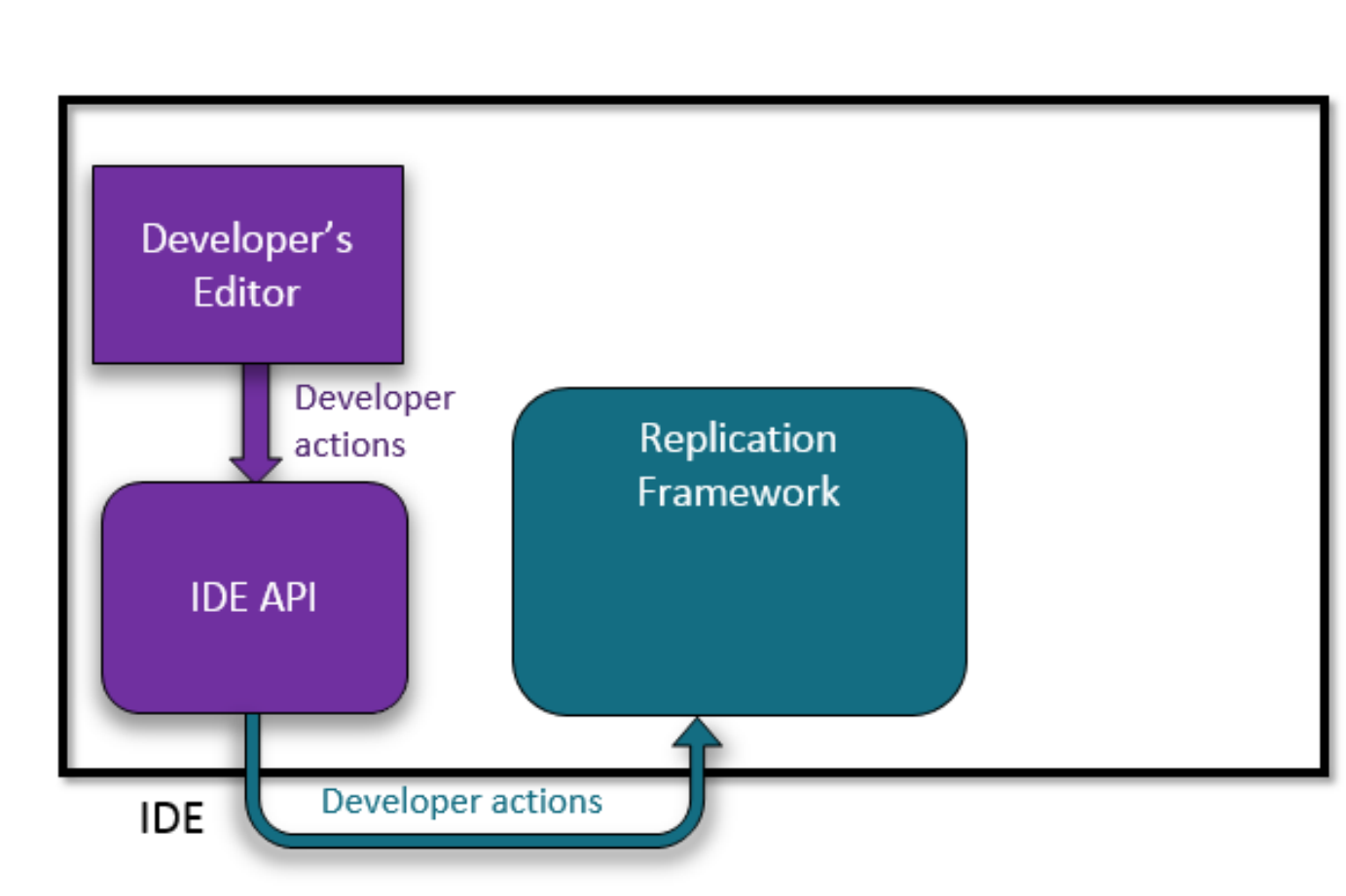
Continuous Analyses

Solution: Copy codebase w/o interference

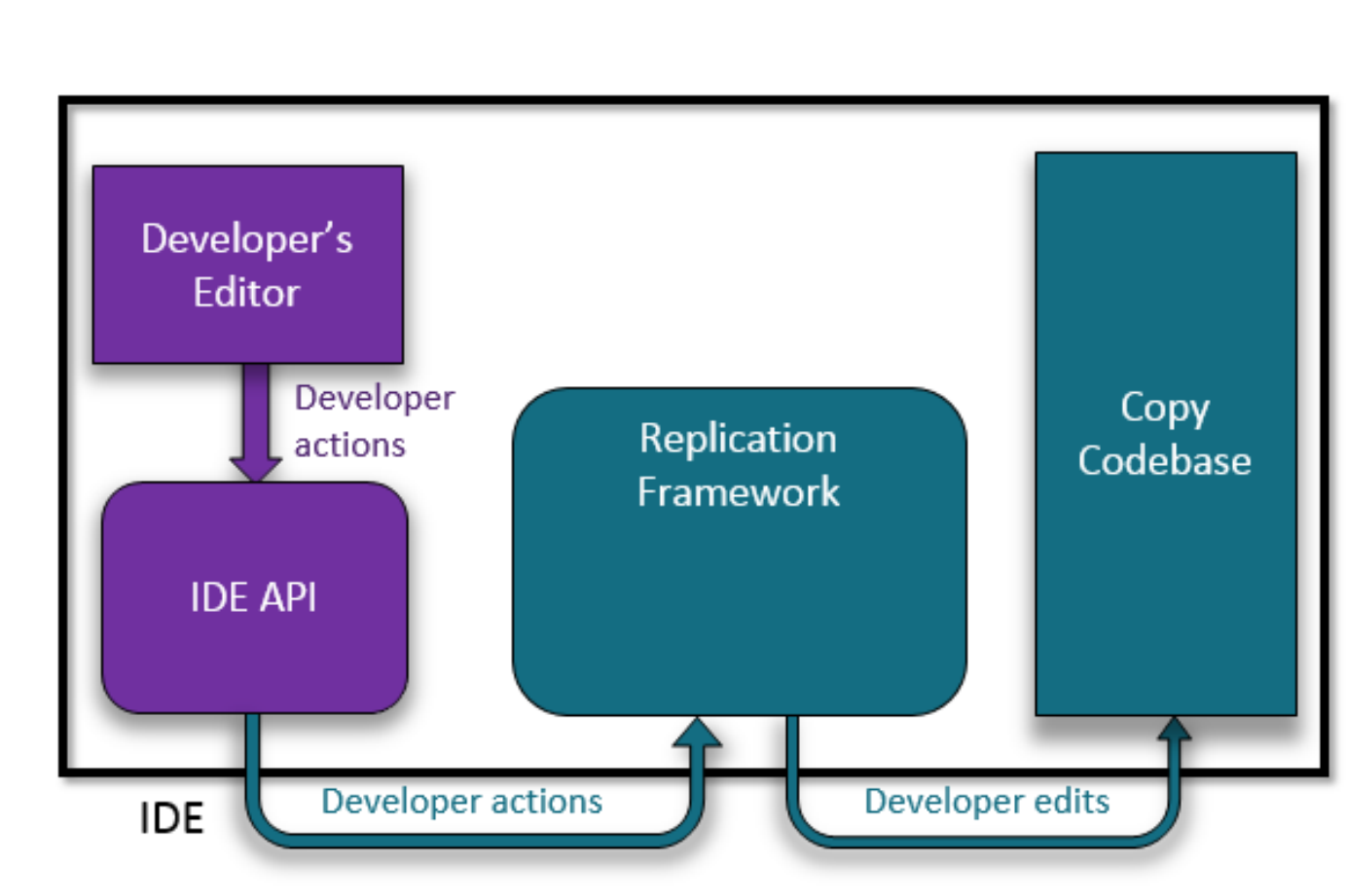
- Wrap non-continuous analyses into continuous ones

* Implemented at Microsoft Research, Summer 2011

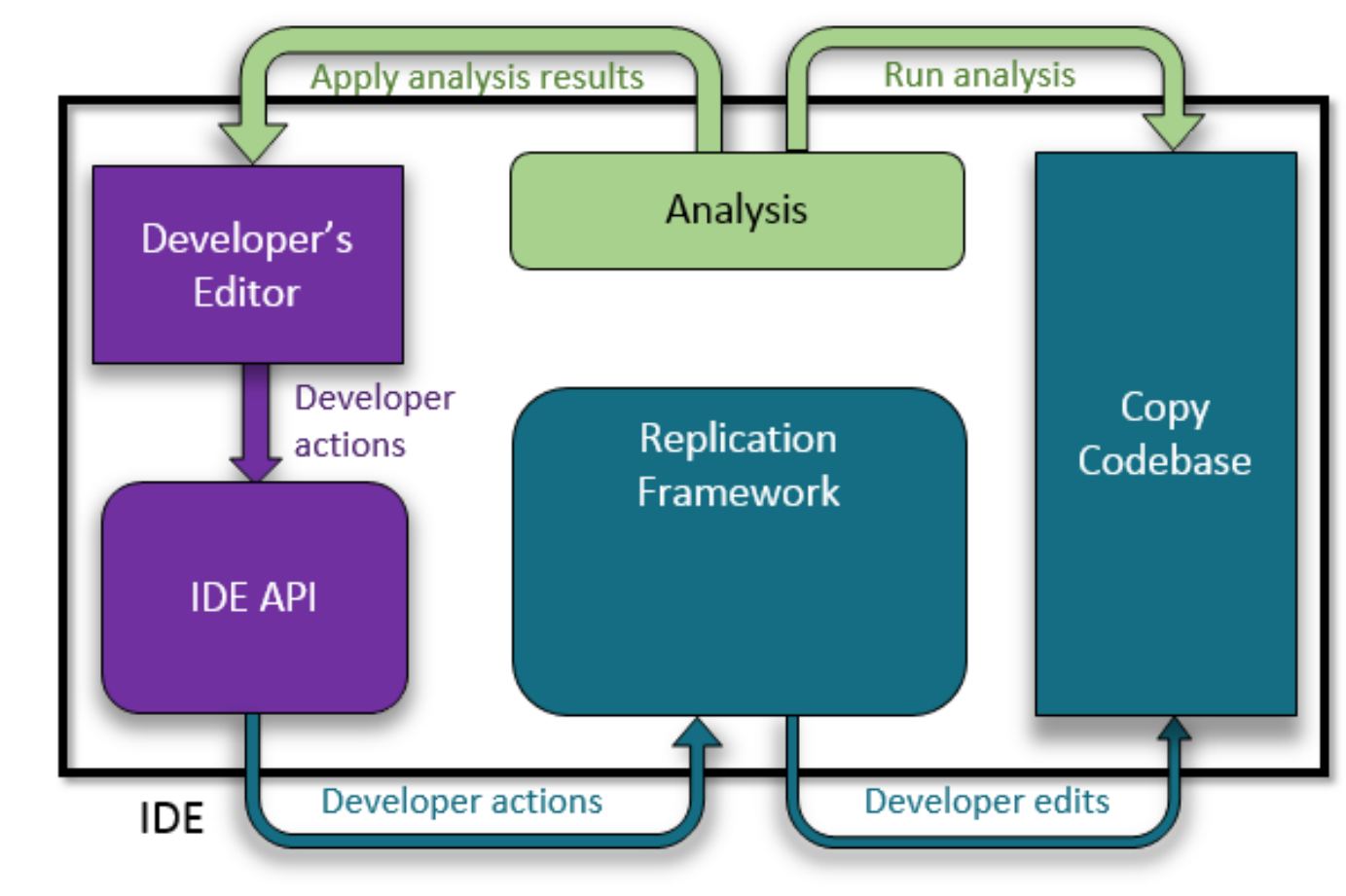
Replication framework: Incremental maintenance of copy codebase



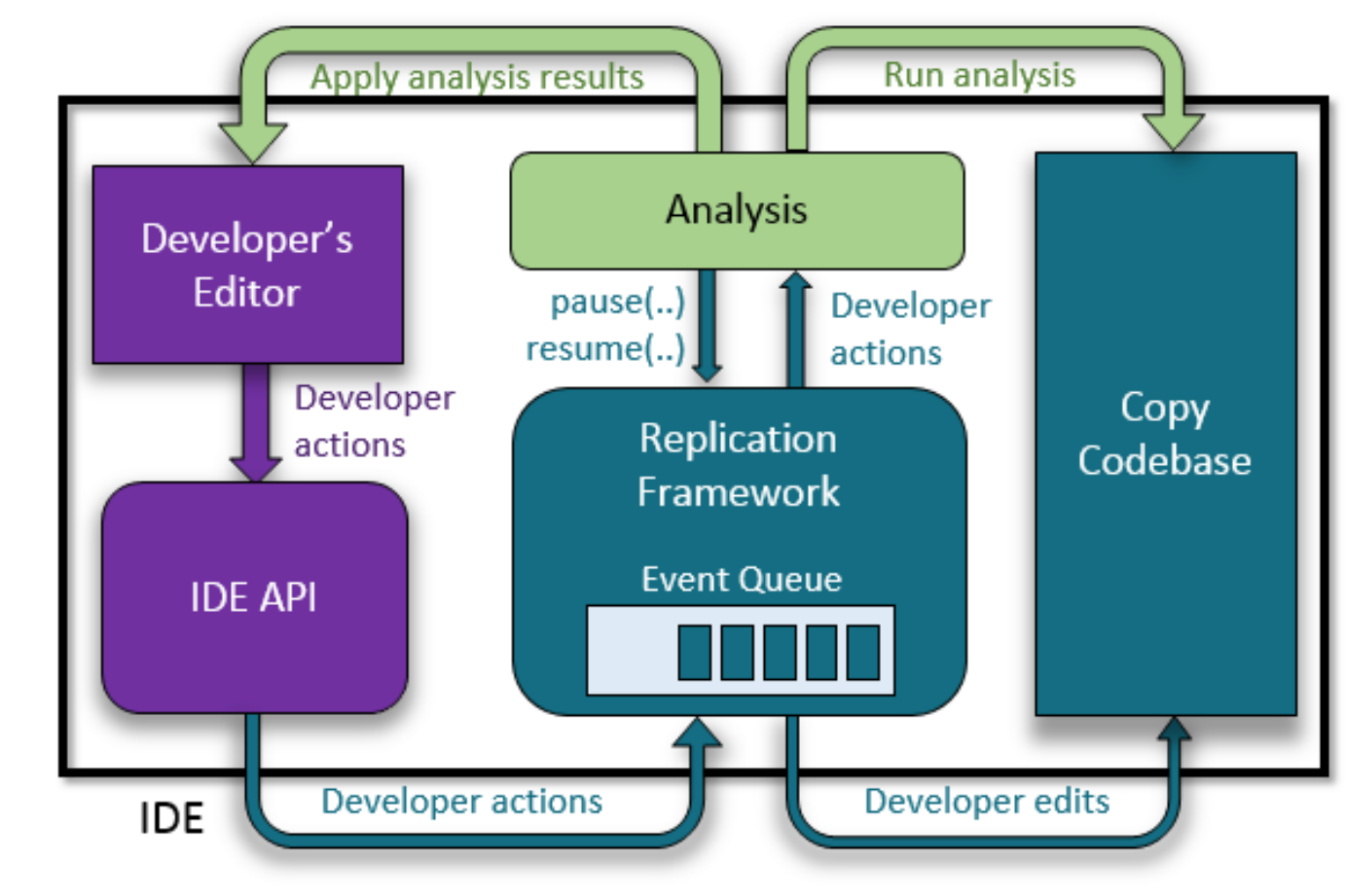
Replication framework: Incremental maintenance of copy codebase



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: ~6 ms / edit

[Avg touch typist, 40 WPM = ~333 ms / edit => ~1.8% overhead]

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

Outline

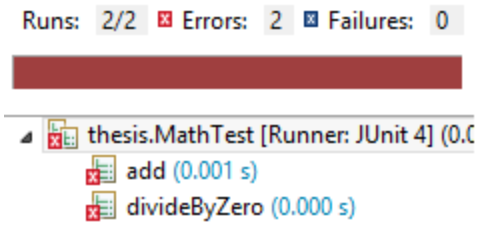
- Motivating example
- Copy codebase and replication framework
  - IDE-integrated continuous analyses
  - Fine-grained development history
    (Required for systematic exploration)
  - Layers: better support for multiple codebases
- Contributions

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");
    }
}
```

2. Fix bug

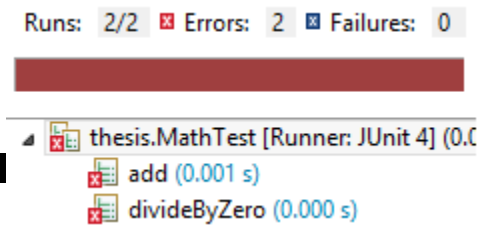
```java
public class MathTest
{
    @Test public void divideByZero() {
        Assert.assertEquals(Double.NaN, Math.divide(1.0, 0.0));
    }

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



Branch: main

Fixing the Bug

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

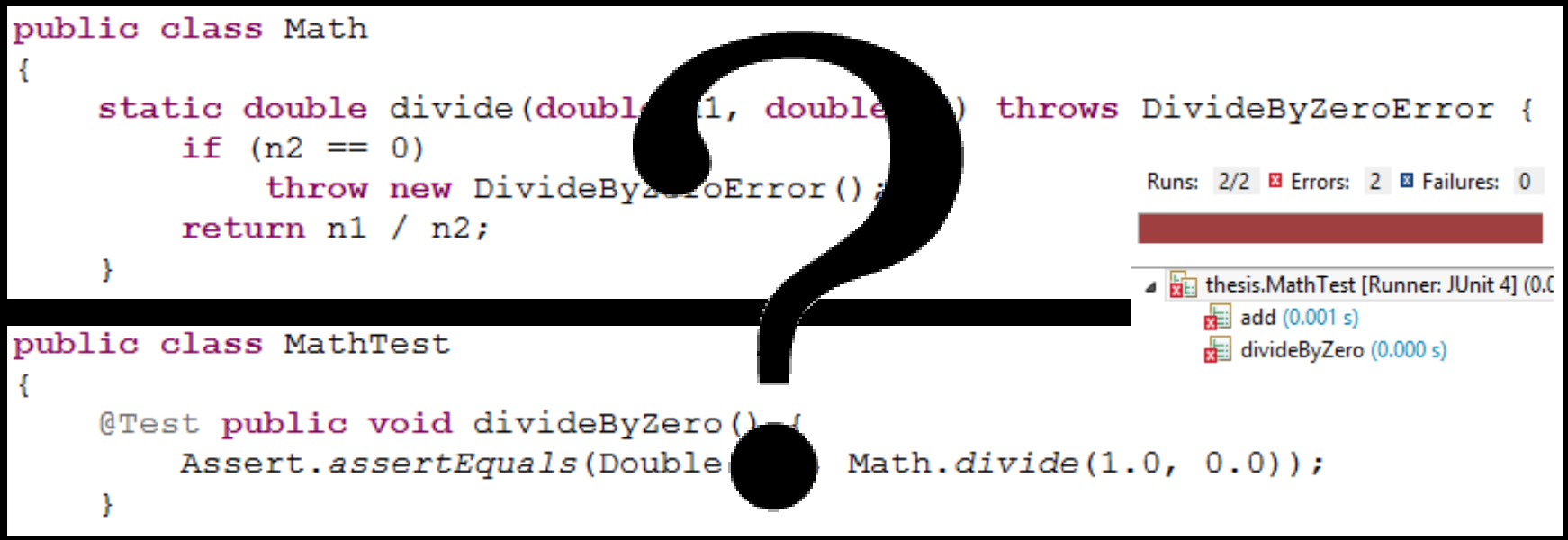



```
public class MathTest
{
    @Test public void divideByZero() {
        Assert.assertEquals(Double.NaN, Math.divide(1.0, 0.0));
    }
```

- 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

Branch: main

Problem 2: Development history MIGHT get lost

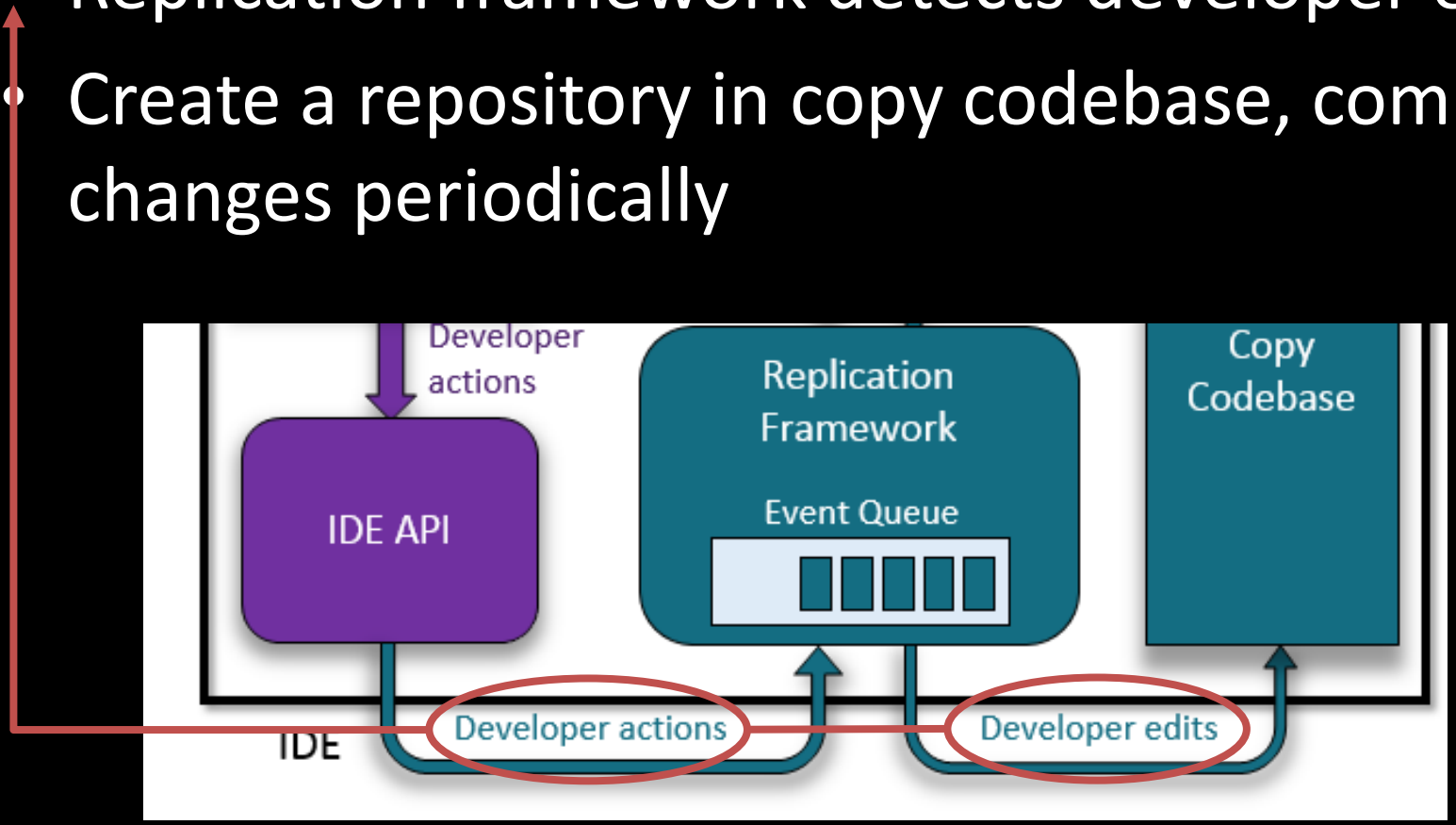
Reason: History management is mostly manual

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]

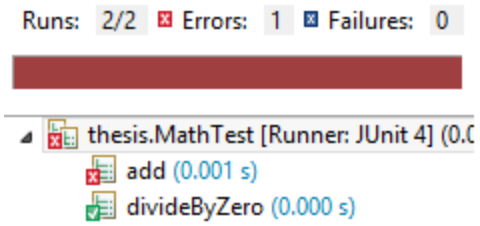
Outline

- 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

```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");
    }
}
```

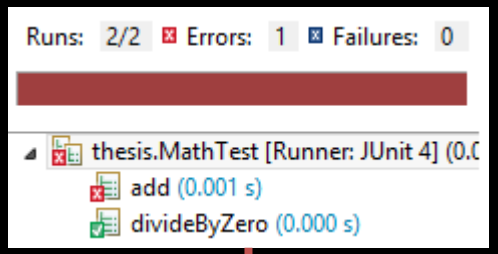
```java
public class MathTest
{
    @Test(expected = DivideByZeroError.class) public void divideByZero() {
        Math.divide(1.0, 0.0);
    }

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



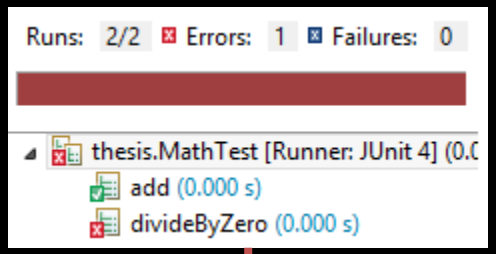
Branch: main

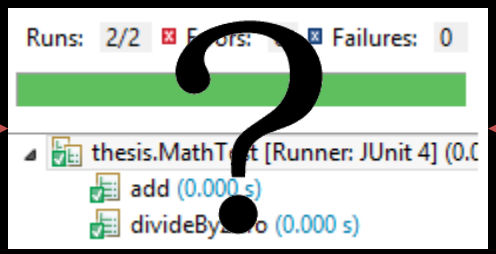
Problem 3: It IS difficult to maintain multiple development codebases

Branch: main



Branch: feature





- 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 [PohlBL 2005]

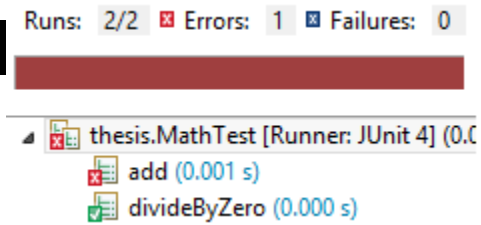
Solution: Scale down Software Product Lines
- Single developer
- Small tasks
- Faster feedback

Layers: parent-child relation between development codebases

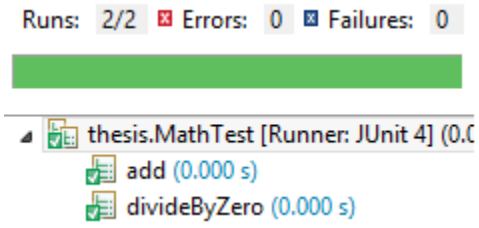
Layers

Branch: main

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

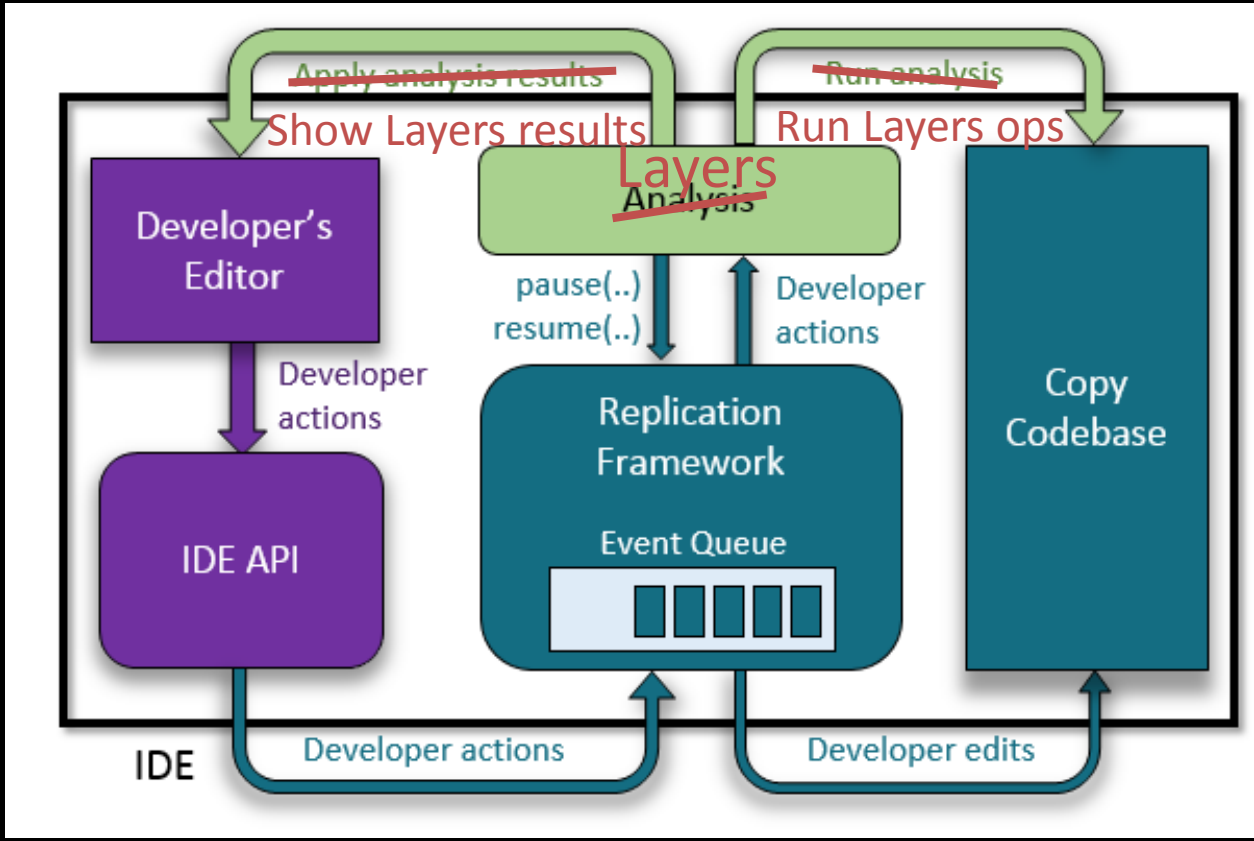



```
public class Math
{
    static double divide(double n1, double n2) throws
        if (n2 == 0)
            throw new DivideByZeroError();
        return n1 / n2;
    }

    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

Layers



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

[1] Beacon. http://blogs.msdn.com/b/msr_er/archive/2011/09/07/seif-project-crystal-receives-acm-sigsoft-distinguished-paper-award.aspx, 2011.

[2] Marcel Bruch, Martin Monperrus, and Mira Mezini. Learning from examples to improve code completion systems. In the 7th joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (FSE), ESEC/FSE'09, pages 213-222, Amsterdam, The Netherlands, August 2009.

[3] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Proactive detection of collaboration conflicts. In the 8th joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (FSE), ESEC/FSE'11, Szeged, Hungary, September 2011.

[4] Zhongxian Gu. Capturing and exploiting fine-grained IDE interactions. In the 34th International Conference on Software Engineering, ICSE'12, pages 1630-1631, Zurich, Switzerland, June 2012.

[5] David Hovemeyer and William Pugh. Finding bugs is easy. In the 19th Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA'04, pages 132-136, Vancouver, BC, CANADA, October 2004.

[6] Mik Kersten and Gail C. Murphy. Mylar: A degree-of-interest model for IDEs. In the 4th International Conference on Aspect-oriented Software Development, AOSD’05, pages 159–168, Chicago, IL, USA, March 2005.

References

[7] Mark Mahoney. The storyteller version control system: Tackling version control, code comments, and team learning. In the 3rd Conference on Systems, Programming, Languages and Applications: Software for Humanity, SPLASH'12, pages 17-18, Tucson, AZ, USA, October 2012.

[8] Kıvanç Muşlu, Yuriy Brun, Michael D. Ernst, and David Notkin. Making offline analyses continuous. In Submission. In the 9th joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (FSE), ESEC/FSE'13, Saint Petersburg, Russia, August 2013.

[9] Kıvanç Muşlu, Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Speculative analysis of integrated development environment recommendations. In the 3rd Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA'12, pages 669-682, Tucson, AZ, USA, October 2012.

[10] Nachiappan Nagappan, Andreas Zeller, Thomas Zimmermann, Kim Herzig, and Brendan Murphy. Change bursts as defect predictors. In the 21st International Symposium on Software Reliability Engineering, ISSRE'10, pages 309-318, San Jose, CA, USA, November 2010.

[11] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., 2005.

[12] David Saff and Michael D. Ernst. Continuous testing in Eclipse. In the 27th International Conference on Software Engineering, ICSE'05, pages 668-669, St. Louis, MO, USA, May 2005.