Making Offline Analyses Continuous

Authors: Kıvanç Muslu, Yuriy Brun, Michael D. Ernst, David Notkin

presenter name(s) removed for FERPA considerations

Introduction

How does a typical programmer write code?

- Write the code and take feedback at compile time

Even if they have separate analysis tools, typically require them to be run manually

- Sit and wait for results before you can begin making more changes

Continuous Analysis

Introduction

Objective:

Make development independent of analysis so that developer workflow is not disturbed.

But how?

By making **offline analyses continuous** using codebase replication.

Definitions

- Offline analysis - requires no user input
- Pure analysis - does not modify the code on which it is running
- Impure analysis - modifies the code on which it is running

Main Contributions

1) Identification of challenges to continuous analysis
2) Codebase replication technique
3) Solstice (eclipse based implementation of codebase replication)
4) Extending offline analysis plug-ins as continuous analysis plug-ins
5) An evaluation on developing continuous analysis plugins for eclipse
6) A case study, testing a solstice based plugin for eclipse

Challenges of Continuous Analysis

- **Isolation**
- **Currency**

Codebase Replication

What is it?

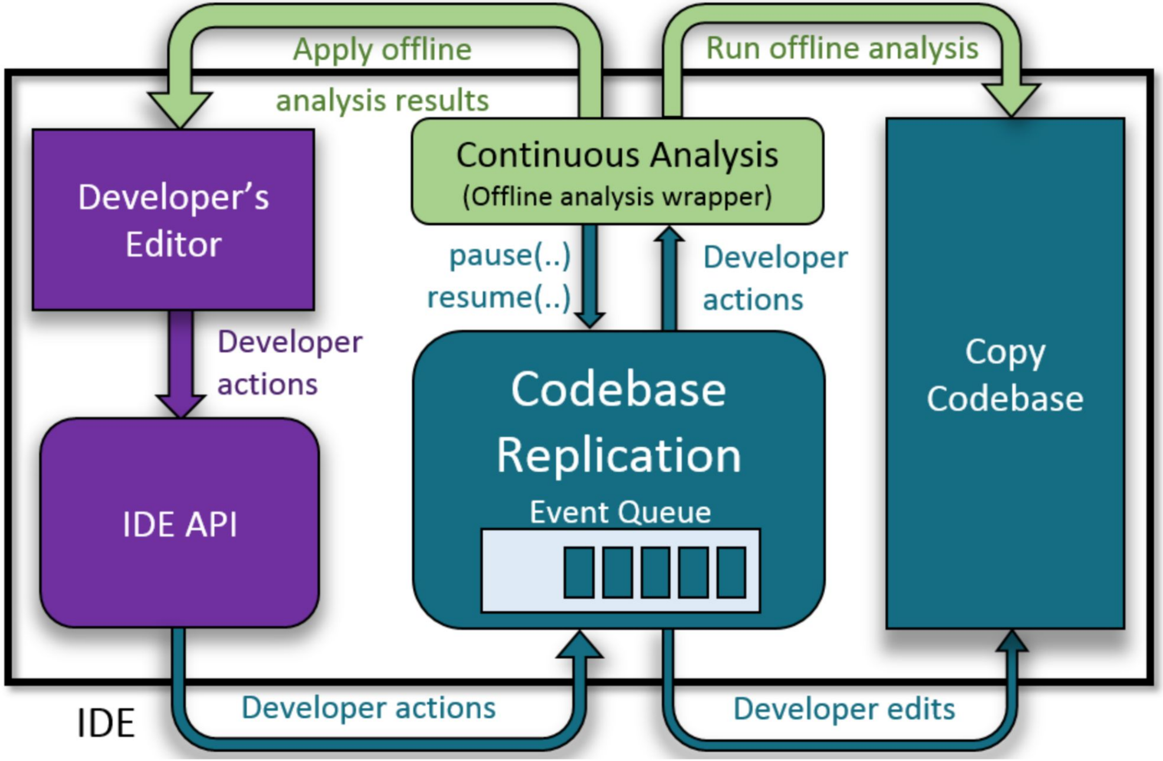
- Creating a copy of a developer’s code

Why is it useful?

- Allows for continuous analysis of code

Technique

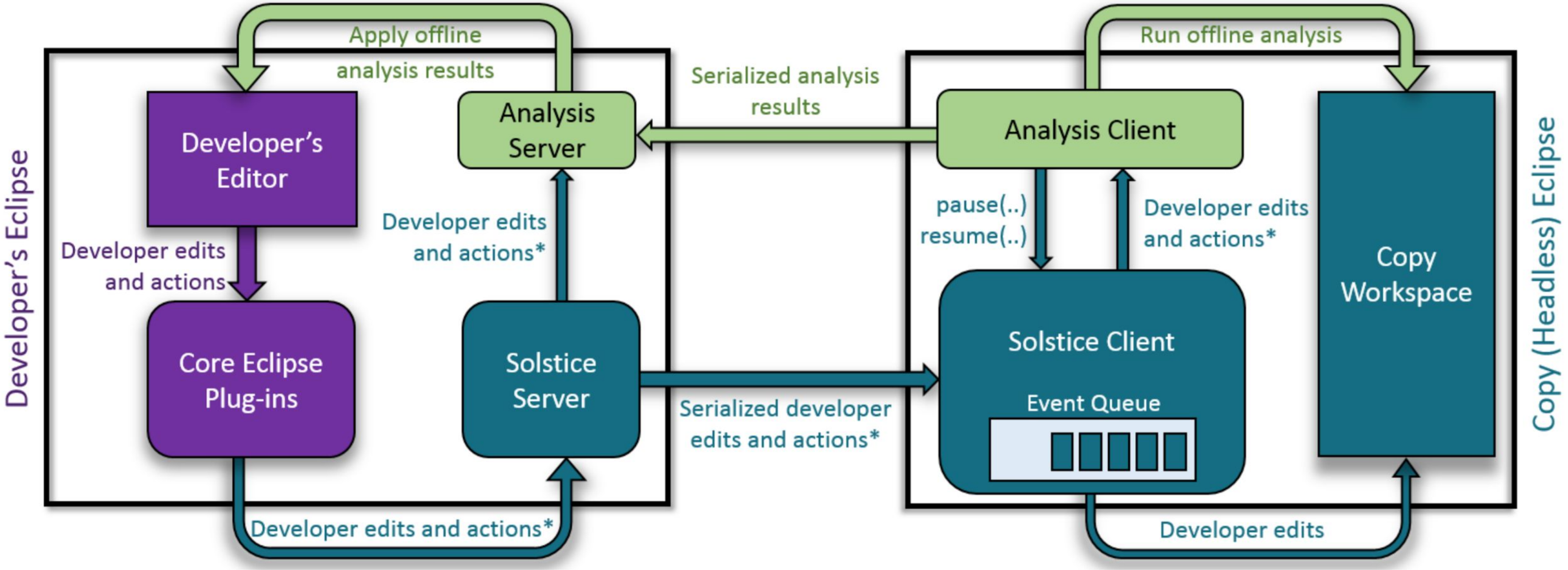
Codebase Replication - A High Level View



Solstice

- An open-source codebase replication prototype for Eclipse
- Allows for authors to create continuous analysis tools (plugins) for Eclipse
- Takes existing codebase replication concept and extends it to a client-server architecture

Solstice Overview



Applying Solstice

- Continuous FindBugs
- Continuous PMD
- Continuous Testing

Continuous Findbugs

Findbugs: A very famous Static Analysis tool for Java programs. Analyses byte code.

Continuous PMD

PMD: Another famous static analysis tool.
Analyses AST generated by JavaCC.

Finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, etc.

Continuous Testing

Uses **idle** CPU cycles to run tests

Can provide a developer with a significant advantage as it notifies them **instantly** when a change breaks their code

Evaluation

Requirements

R1: The start up time should not block the developer for an reasonable amount of time.

R2: The overhead to synchronizing developer actions with the copy codebase should be close to zero.

R3: While using the IDE, the developer should experience negligible delay in performing actions.

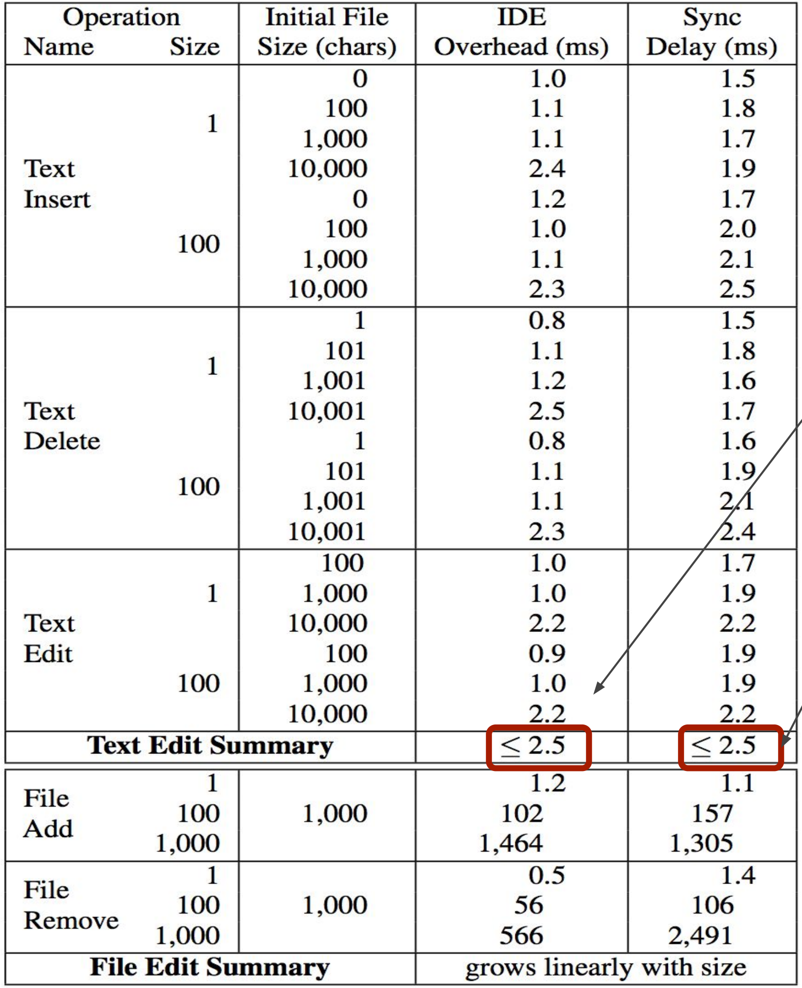
Performance

Low **overhead** and low **delay**

Average sync times:

- Full sync
    - 1.3 MB File = 131 ms
    - 176 MB File = 2.9 min

- Incremental sync
    - 1.3 MB File = 133 ms
    - 176 MB File = 6.9 s



Operation Name	Size	Initial File Size (chars)	IDE Overhead (ms)	Sync Delay (ms)
Text Insert	1	0	1.0	1.5
		100	1.1	1.8
		1,000	1.1	1.7
		10,000	2.4	1.9
	100	0	1.2	1.7
		100	1.0	2.0
		1,000	1.1	2.1
		10,000	2.3	2.5
Text Delete	1	1	0.8	1.5
		101	1.1	1.8
		1,001	1.2	1.6
		10,001	2.5	1.7
	100	1	0.8	1.6
		101	1.1	1.9
		1,001	1.1	2.1
		10,001	2.3	2.4
Text Edit	1	100	1.0	1.7
		1,000	1.0	1.9
		10,000	2.2	2.2
	100	100	0.9	1.9
		1,000	1.0	1.9
		10,000	2.2	2.2
Text Edit Summary			≤ 2.5	≤ 2.5
File Add	1	1,000	1.2	1.1
	100		102	157
	1,000		1,464	1,305
File Remove	1	1,000	0.5	1.4
	100		56	106
	1,000		566	2,491
File Edit Summary			grows linearly with size	

Usability

On average, writing the three proof of concepts (continuous findbugs, continuous PMD, continuous testing) required:

- ~ 519 LOC
- ~ 18 Hours to write
- 2.5 ms overhead
- 2.5 ms syncing delay

Review:

- What is the scientific question? the answer?
- What’s the key new idea that allows answering it?
- How do you measure the success of the answer?

Discussion

Discussion Questions

Question 1:

IDE based continuous analysis systems are not new.
What makes Solstice unique?

Discussion Questions

Question 2:

How could this technique scale to a multi-developer model?

Discussion Questions

Question 3:

If Solstice is deployed on a distributed system, what problems can arise?

Discussion Questions

Question 4:

How does Solstice know whether to terminate immediately or keep running the analysis upon detecting stale results? Is this a configuration setting?

Discussion Questions

Question 5:

Could there be any potential issues with resource usage (specifically with disk space)?

Discussion Questions

Question 6:

What are some possible optimizations that could be made to Solstice?

Sources

- Making Offline Analyses Continuous