CSE 503

Software Engineering

Course introduction

Key questions:
What does your program do?
How do you know?

Logistics

- Tue/Thu, 11:30am – 12:50pm
- Course material, schedule, etc. on course website:
  https://courses.cs.washington.edu/courses/cse503/
  All slides are posted before class.
- Assignment submission and discussions via Canvas (linked from webpage)
- Instructor: Michael Ernst
  - Office hours: After class and by appointment
  - mernst@cs.washington.edu

Logistics

- Tue/Thu, 11:30am – 12:50pm.
- Course material, schedule, etc. on course website:
  https://courses.cs.washington.edu/courses/cse503/
  All slides are posted before class.
- Assignment submission and discussions via Canvas (linked from webpage)

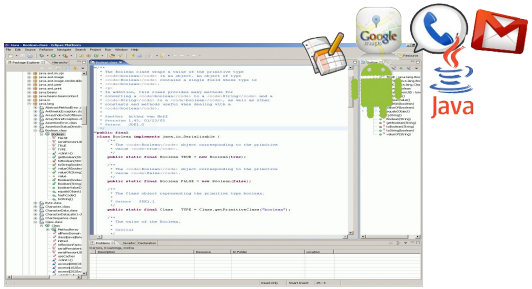
Today

- Course overview & expectations
- Brief introductions
- Why program analysis?

What is Software Engineering?

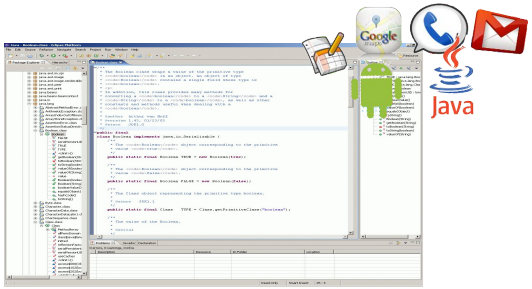
What is Software Engineering?

- Developing in an IDE and software ecosystem



What is Software Engineering?

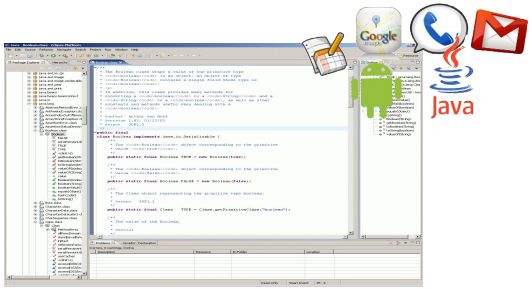
- Developing in an IDE and software ecosystem

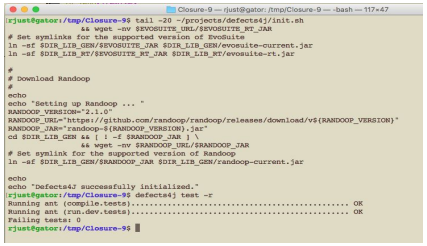


- Testing and debugging

What is Software Engineering?

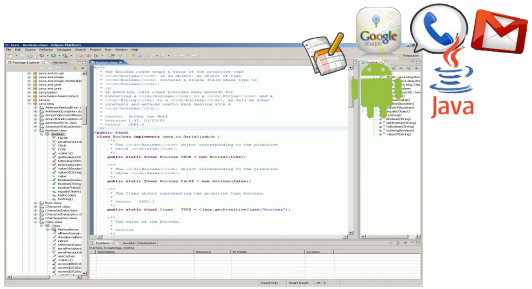
- Developing in an IDE and software ecosystem
- Testing and debugging
- Deploying and running a software system

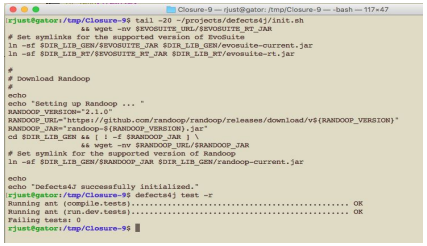


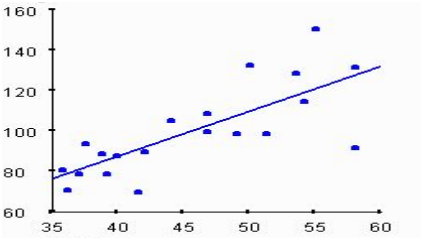


What is Software Engineering?

- Developing in an IDE and software ecosystem
- Testing and debugging
- Deploying and running a software system
- Empirical evaluations

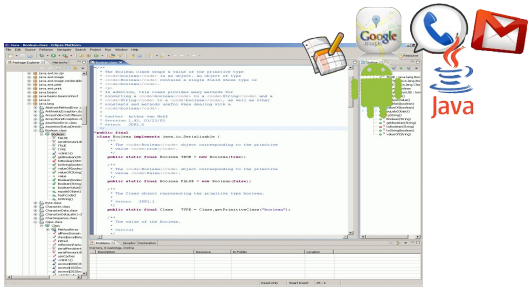


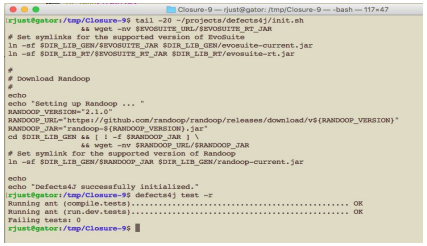


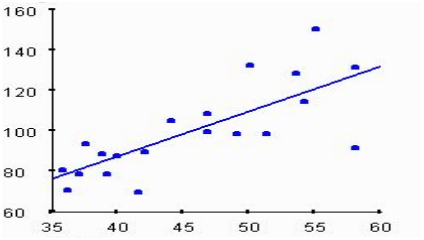


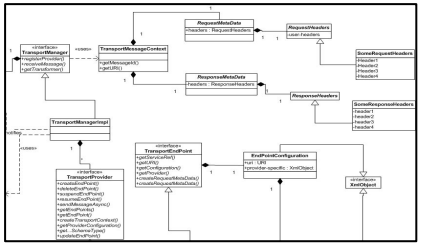
What is Software Engineering?

- Developing in an IDE and software ecosystem
- Testing and debugging
- Deploying and running a software system
- Empirical evaluations
- Modeling and designing









What is Software Engineering?

More than just writing code

The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring

What is Software Engineering?

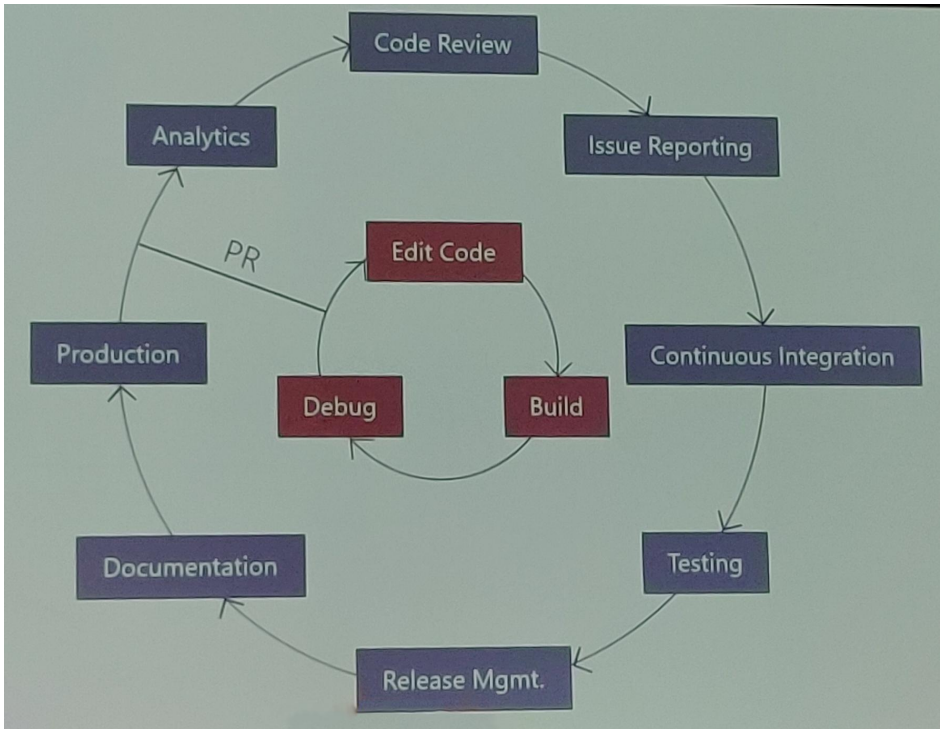
More than just writing code

The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring

Just one out of many important tasks!

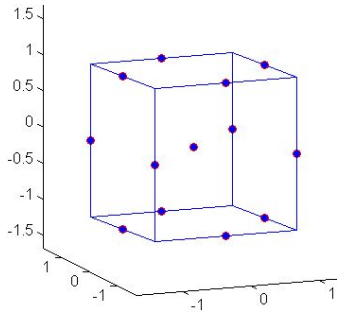
The Role of Software Engineering in Practice

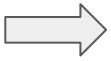


(Development workflow at Microsoft, Big Code summit 2019)

The Role of Software Engineering in Research

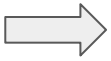
Experimental infrastructure is software, too!





Infrastructure

Design space exploration



1	0.34	0.81
2	0.52	0.32
3	0.21	0.53
4	0.81	0.22
...	...	...

Example (automated debugging)

- 150 configurations, 1000+ benchmarks
- 1-85 hours per execution
- 200,000+ CPU hours (~23 CPU years)

Course overview: the big picture

- **Week 1:** Introduction; static vs. dynamic analysis
- **Week 2:** Symbolic reasoning
- **Week 3:** Symbolic reasoning
- **Week 4:** Testing
- **Week 5:** Delta Debugging
- **Week 6:** Invariants
- **Week 7:** Program Repair
- **Week 8:** Empirical Software Engineering
- **Week 9:** ML for Software Engineering
- **Week 10:** Wrap up

Course overview: the big picture

- **Week 1:** Introduction; static vs. dynamic analysis — **HW 1**
- **Week 2:** Symbolic reasoning
- **Week 3:** Symbolic reasoning — **HW 2**

formal, static

- **Week 4:** Testing
- **Week 5:** Delta Debugging
- **Week 6:** Invariants — **In-class exercise**

depends on executions

- **Week 7:** Program Repair
- **Week 8:** Empirical Software Engineering
- **Week 9:** ML for Software Engineering

depends on heuristics

- **Week 10:** Wrap up — **Project presentation**

Questions?

Course overview: this week

- **Week 1:** Introduction & static vs. dynamic analysis **HW 1**

- **Two high-level papers**
  - Static and dynamic analysis: synergy and duality
  - Lessons from building static analysis tools at Google
- **HW 1**
  - Brainstorming about software development difficulties
  - **Please** start right away!

Course overview: the project

Logistics

- 2-4 team members
- Synergies with **your** work are welcome!

Timeline

- **Week 3/4**: Project proposal and revision
- **Week 6**: Related work and methodology
- **Week 8**: Coding completed and initial results
- **Week 10**: Presentation and final report

Course overview: the project

Logistics

- 2-4 team members
- Synergies with **your** work are welcome!

Timeline

- **Week 3/4**: Project proposal and revision
- **Week 6**: Related work and methodology
- **Week 8**: Coding completed and initial results
- **Week 10**: Presentation and final report

Types of projects (non-exhaustive)

- proposing and evaluating a new technique
- developing and assessing new algorithms to replace currently-used ones
- translating a methodology to a new problem domain
- applying known techniques to new problem domains
- evaluation of existing techniques or tools (case studies or controlled experiment)
- implementation of a proposed but never implemented technique

Questions?

Your course project might be publishable

Combining Dynamic and Static Analyses to Recover Object-Oriented Features from C++ Binaries

Matthew Arnold Mark Polyakov Michael D. Ernst
University of Washington
Seattle, WA, USA
mma35@uw.edu *xe@uw.edu* *mernst@cs.uw.edu*

***Abstract*—We present a new hybrid static–dynamic analysis technique to discover object-oriented features (such as classes and methods) in binaries compiled from C++ source code. The dynamic analysis records method calls whose first argument appears to be an object pointer. The static analysis discovers more procedures that must take the same first argument as an executed method.**

We implemented our te named KreO. We compared dynamic analysis (Lego) and discovering the same object-

KreO tends to have hi Lego. KreO tends has lower OOAnalyzer; however, Kre OOAnalyzer for code that is

1. Introduction

State-of-the-art decompi Pro [2] can generate C-like When an executable is comp to discover classes, metho collectively term these as ' Such information helps an the software and removes

The developer can use KreO to find private methods on that class, or find methods on parent classes, which the developer may wish to call.

Existing tools for detecting OO features use a variety of static and dynamic analyses. The state-of-the-art static analysis is OOAnalyzer [7]. OOAnalyzer extracts low-level facts about a binary using simple heuristics and dataflow

Evaluation of Version Control Merge Tools

Anonymous Author(s)

ABSTRACT

A version control system, such as Git, requires a way to integrate changes from different developers or branches. A merge tool automatically integrates some changes, deferring others for manual resolution. The dominant line-based merge tools, such as Git Merge, leave many cases for manual resolution, which consumes valuable developer time.

Most evaluations of merge tools suffer from three main problems: they use flawed methods to determine merge correctness, they are not evaluated on representative merges, and they do not compare with state-of-the-art tools.

1.1 Merge correctness

And there is more...

Special topics:

- **504: AI meets Software engineering**
  (ML and statistical methods for SE/program analysis)
- **599: Research methods**
  (Research design and statistics in R)

Course overview: the big picture

- **Week 1:** Introduction; static vs. dynamic analysis **HW 1**
- **Week 2:** Symbolic reasoning
- **Week 3:** Symbolic reasoning **HW 2**
- **Week 4:** Testing
- **Week 5:** Delta Debugging **In-class exercise**
- **Week 6:** Invariants
- **Week 7:** Program Repair
- **Week 8:** Empirical Software Engineering
- **Week 9:** ML for Software Engineering
- **Week 10:** Wrap up **Project presentation**

Course overview: grading

- **50%** Class project
- **35%** HWs, in-class exercise, reading questions
- **15%** Participation

Your activities each week:

- Read papers
- Participate in class
- Make progress on your research project
- Submit writeup of your project

Questions?

Course overview: expectations

- Some programming experience
- Conducting a quarter-long research project
- Reading and actively discussing research papers
- Have fun!

Who can be successful?

- **You** can!
- Assumes an undergraduate CS education (~ 1st year grad)
- You will *learn* to read papers, write papers, conduct research, etc. That is a goal of the class.
- Ask lots of questions, so we can help you

Course overview: the big picture

- **Week 1:** Introduction; static vs. dynamic analysis **HW 1**
- **Week 2:** Symbolic reasoning
- **Week 3:** Symbolic reasoning **HW 2**
- **Week 4:** Testing
- **Week 5:** Delta Debugging **In-class exercise**
- **Week 6:** Invariants
- **Week 7:** Program Repair
- **Week 8:** Empirical Software Engineering
- **Week 9:** ML for Software Engineering
- **Week 10:** Wrap up **Project presentation**

If CSE 503 is not enough for you...

Special topics:

- **CSE 590N: LLMs and code**
  (ML and statistical methods for software engineering)
  Mondays 3:30-4:20

Today

- Course overview & expectations
- Brief introductions
- Why program analysis?

The CSE 503 team

Instructor

- Michael Ernst
- Office hours: After class and by appointment
- mernst@cs.washington.edu

Teaching assistant

- Thomas Schweizer
- Office hours: TBD
- tschweiz@cs.washington.edu

Instructor

Michael Ernst

- Office hours: After class and by appointment
- mernst@cs.washington.edu

Your background

Introduction and a very brief survey

- What is your research area (or area of interest)?
- How long have you been in the program?
- What is your SE background (programming languages, etc.)?

Today

- Course overview & expectations
- Brief introductions
- Why program analysis?

Who cares about program analysis?

Who cares about program analysis?

- ~15 million lines of code

Let’s say 50 lines per page (0.05 mm)

Who cares about program analysis?

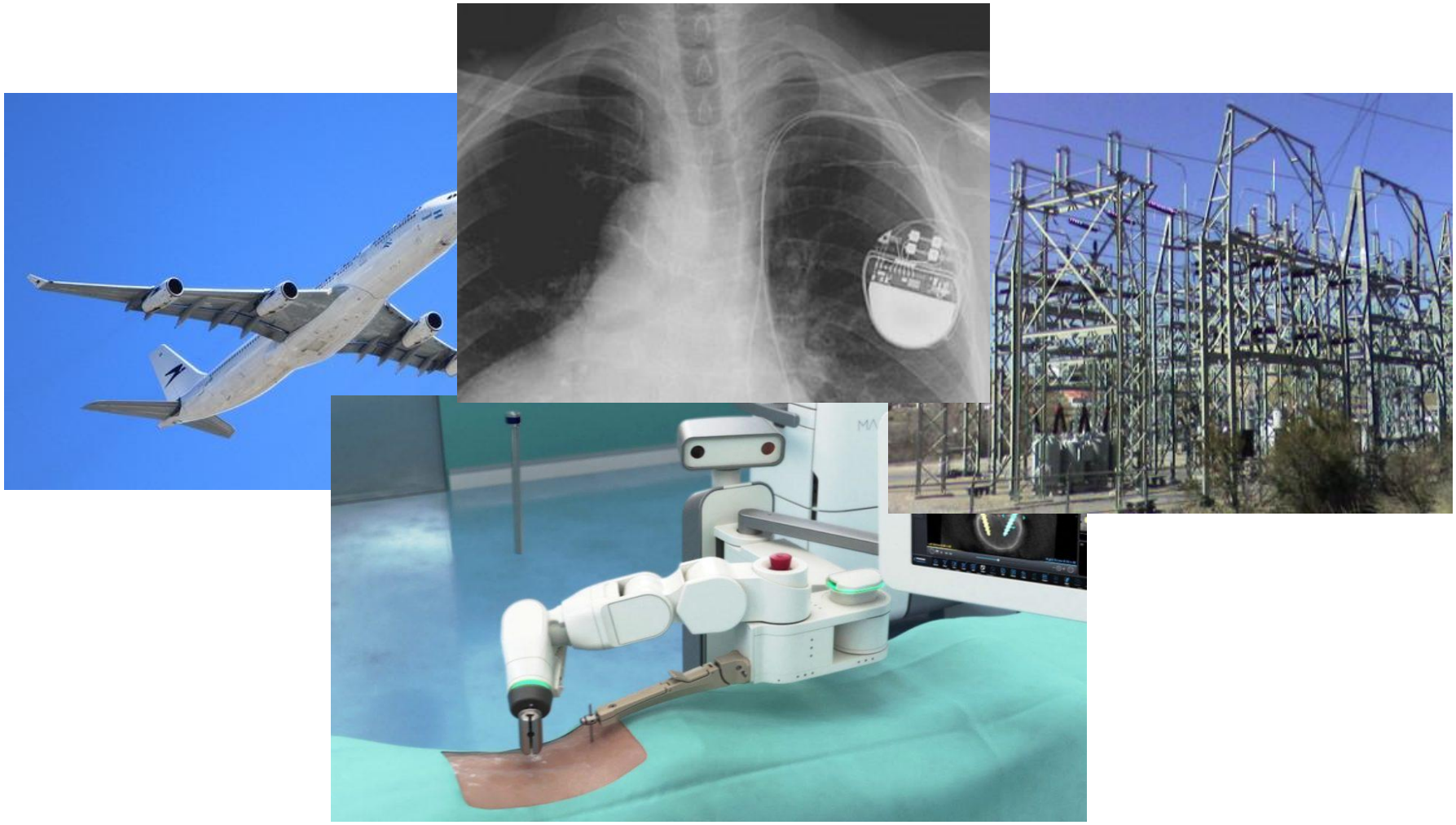
~15 million lines of code

At 50 lines per page:

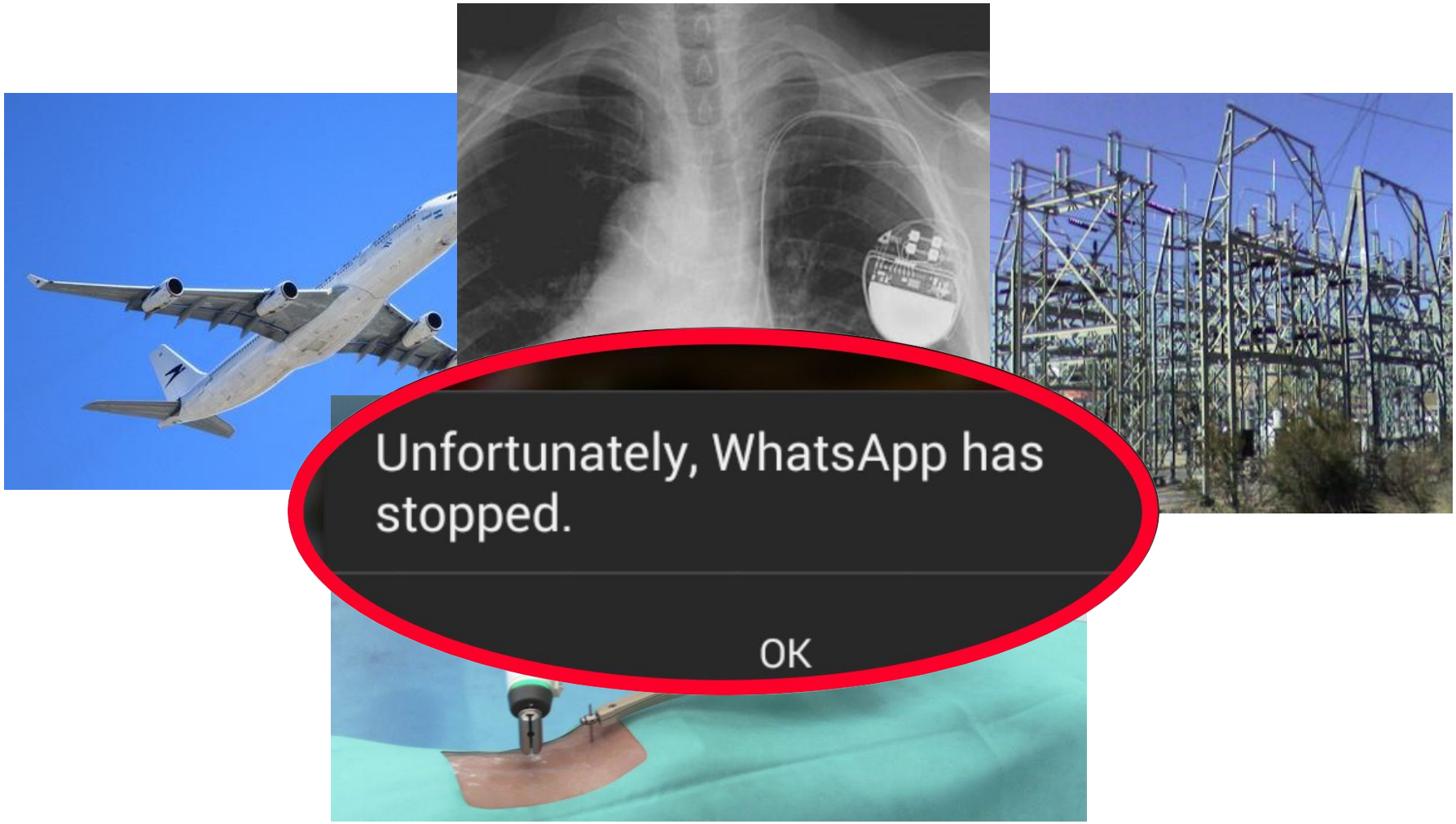
- 300000 pages
- 15 m (49 ft)

Software cost > airframe

Who cares about program analysis?



Who cares about program analysis?



Program analysis: examples

Does my program implement its specification?

```
double foo(double[] d {
  int n = d.length;
  double s = 0;
  int i = 0;
  while (i<n)
  s = s + d[i];
  i = i + 1;
  double a = s / n;
  return a;
}
```

	Business Requirements Document (BRD)	Software Requirement Specifications (SRS)	Functional Requirement Specifications (FRS)
Other names		Product Requirements Document (PRD) and System Requirements Specification	Functional Specifications Document (FSD), Product Specification Document (PSD), Functional Specs (FS)
Created By	Business Analyst	Business/System Analyst	Business/System Analyst/Implementation Leads
Contains	High level business requirements and stakeholder requirements	Detailed functional requirements, non-functional requirements and use cases	Granular functional requirements, data flow and UML diagrams
Used By	Upper and middle management	Project managers, SMEs (subject matter experts), technical and implementation lead	Technical leads, development teams and testing teams.
Prepared in	Initiation phase	Planning phase	Planning phase
Answers	'Why' the requirements are being undertaken	'What' requirements must be fulfilled to satisfy business needs	'How' exactly the system is expected to function
Example	Improve efficiency by tracking the employee time in office	Proposed software will contain following modules: Login, Administrator, Employee and Reporting	Login module will contain fields like: Enter username, Enter password, Submit button

Program analysis: examples

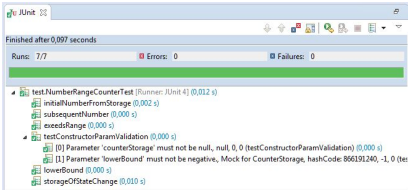
Does my program implement its specification?

```
double foo(double[] d {
  int n = d.length;
  double s = 0;
  int i = 0;
  while (i<n)
  s = s + d[i];
  i = i + 1;
  double a = s / n;
  return a;
}
```

	Business Requirements Document (BRD)	Software Requirement Specifications (SRS)	Functional Requirement Specifications (FRS)
Other names		Product Requirements Document (PRD) and System Requirements Specification	Functional Specifications Document (FSD), Product Specification Document (PSD), Functional Specs (FS)
Created By	Business Analyst	Business/System Analyst	Business/System Analyst/Implementation Leads
Contains	High level business requirements and stakeholder requirements	Detailed functional requirements, non-functional requirements and use cases	Granular functional requirements, data flow and UML diagrams
Used By	Upper and middle management	Project managers, SMEs (subject matter experts), technical and implementation lead	Technical leads, development teams and testing teams.
Prepared in	Initiation phase	Planning phase	Planning phase
Answers	'Why' the requirements are being undertaken	'What' requirements must be fulfilled to satisfy business needs	'How' exactly the system is expected to function
Example	Improve efficiency by tracking the employee time in office	Proposed software will contain following modules: Login, Administrator, Employee and Reporting	Login module will contain fields like: Enter username, Enter password, Submit button

Example analyses

- Unit testing



- Solver-aided reasoning

$$(\forall x\ \mathrm{fsa}(x)) \Rightarrow (\exists y\ \mathrm{pda}(y) \wedge \mathrm{equivalent}(x, y))$$

Program analysis: examples

What does this program (binary) do?

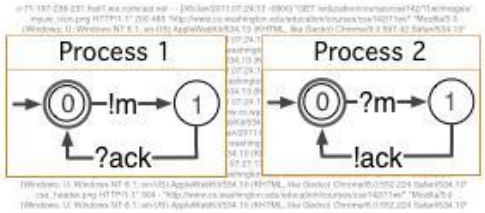
Program analysis: examples

What does this program (binary) do?

Example analyses

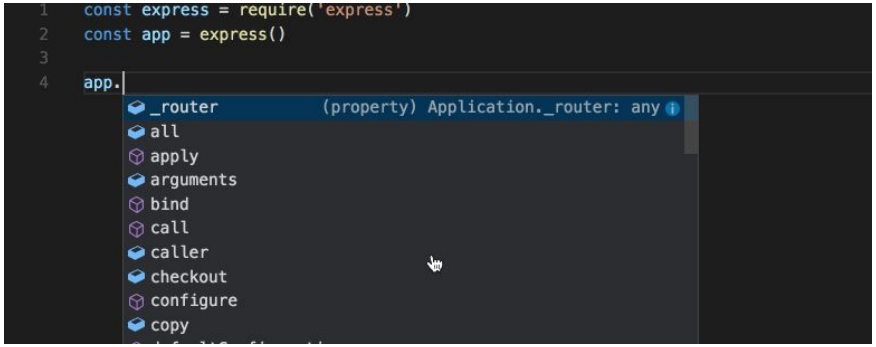
- Fuzzing

- Statistical inference of invariants and models



Program analysis: examples

Autocompletion: which methods to suggest?



Program analysis: examples

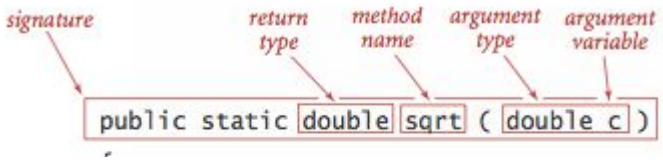
Autocompletion: which methods to suggest?

```
1  const express = require('express')
2  const app = express()
3
4  app.
```

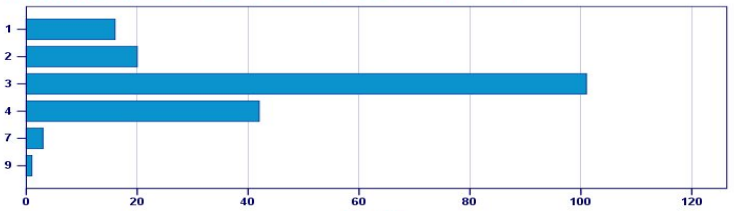
```
_router          (property) Application._router: any
all
apply
arguments
bind
call
caller
checkout
configure
copy
```

Example analyses

- Context-sensitive type checking



- Heuristics and frequency analysis



Program analysis: examples

Semantics: how to name this method?

```
void f(int[] array) {
    boolean swapped = true;
    for (int i = 0; i < array.length && swapped; i++) {
        swapped = false;
        for (int j = 0; j < array.length - 1 - i; j++) {
            if (array[j] > array[j+1]) {
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1]= temp;
                swapped = true;
            }
        }
    }
}
```

Program analysis: examples

Semantics: how to name this method?

```
void f(int[] array) {
    boolean swapped = true;
    for (int i = 0; i < array.length && swapped; i++) {
        swapped = false;
        for (int j = 0; j < array.length - 1 - i; j++) {
            if (array[j] > array[j+1]) {
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1]= temp;
                swapped = true;
            }
        }
    }
}
```

Example analyses

- Statistical language models
  (bag of words, n-grams, etc.)

- Machine learning (LLMs, …)

Next time: static vs. dynamic analysis

A **static analysis** analyzes program source code without running the program

- What are examples?

A **dynamic analysis** observes program executions

- What are examples?

Under what circumstances is each one preferable?