Software Testing, Quality Assurance, and Maintenance (STQAM)

ECE 653

Lecture 1: Tuesday, 07.05.2019

Werner Dietl

https://ece.uwaterloo.ca/~wdietl/

UNIVERSITY OF WATERLOO

Instructor and TA

Instructor

Prof. Werner Dietl wdietl@

Teaching Assistants

None so far

Course Web Page

LEARN: https://learn.uwaterloo.ca

Course Time and Location

Date: Tuesday Time: 11:30 – 14:20

Location: E7 5353

No tutorials

Office hours will be by appointment

Begin all email subjects with **[ECE653]**

Identify yourself

- Originated from your uwaterloo email address, or
- Signed with your full name and student ID

Grading

Assignments: 30%

Quizzes: 20%

Final Exam: 50%

Grades may be curved or adjusted at the Instructor's discretion

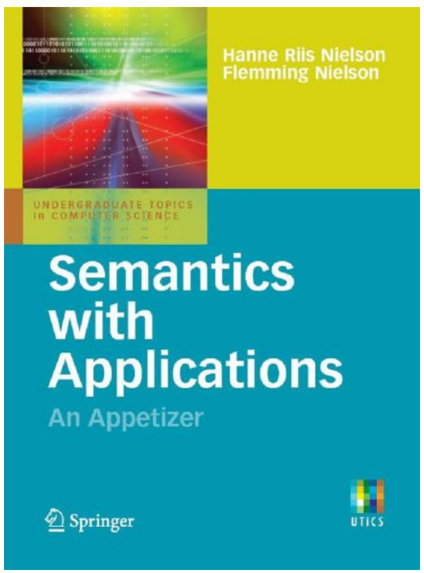
1 + 3 Assignments

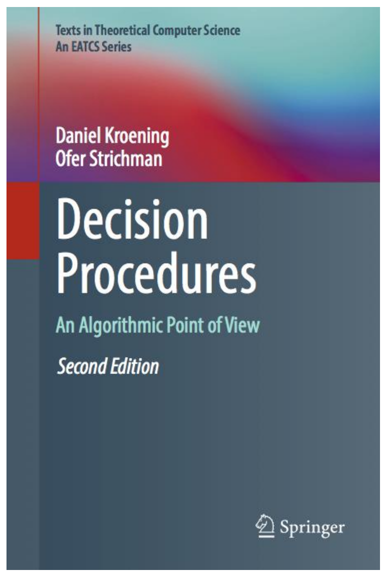
- Pen and paper exercises, and
- Programming assignments
  - mostly in Python

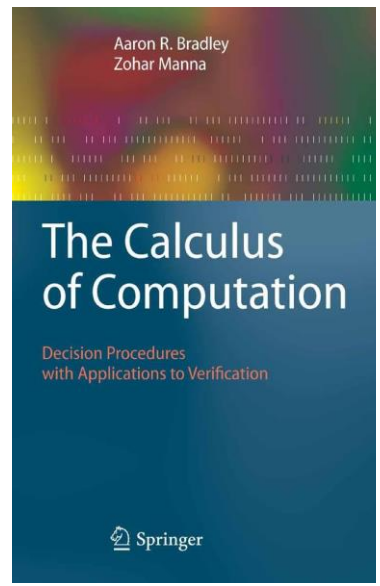
Textbook and Lecture Notes

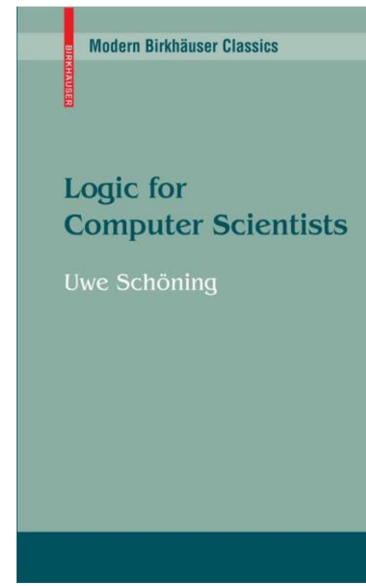
No required text book. Lecture slides and notes will be provided.

- LEARN: https://learn.uwaterloo.ca









Course Website & LEARN

LEARN is the definitive source

- When in doubt, consult LEARN
- Check syllabus for final grade computation

YOUR responsibility to check for updates!

- LEARN (http://learn.uwaterloo.ca)

GitHub

We will use **GitHub** for managing and submitting assignments

- This requires a free GitHub account
- Follow the link in Assignment 0 to get started
- Let me know if there are any problems!!!
- GitHub Tutorial: https://try.github.io

UNIVERSITY OF WATERLOO

Independent Work

All work turned in must be of that individual student unless stated otherwise.

Violations will result in zero credit to all students concerned. University of Waterloo Policy 71 will be followed for any discovered cases of plagiarism.

Policy on Late Assignments

You have 2 days of lateness for assignments that you can use throughout the term

- These are TWO days for the term. Not for each assignment!

Each day the assignment is late consumes one day of lateness

For example,

- You can be 2 days late on assignment A1, or
- One day late on A1, and one day late on A3, or
- You can hand all of the assignments on time ☺

UNIVERSITY OF WATERLOO

Missed Quiz

If you miss a quiz, you will receive 0 for it. If you have a legitimate reason (at the discretion of the instructor) for not taking the quiz and obtain a permission from the instructor **a week** in advance, the percentage for the quiz will be shifted to the final.

See syllabus for more details.

Is this course for me?

Not a TESTING course!

- Foundations of Testing / Coverage
- Foundations of Symbolic Execution and Symbolic Reasoning
- Foundations of Deductive Program Verification
- (Possibly) Foundations of Automated Verification

Enough background?

- Can you code? (Python?) https://docs.python.org/2.7/tutorial/
- Have you used a Unix/Linux machine before?
  - command line, shell, editor…
- Do you know Logic / Automated Reasoning?
  - Propositional logic: AND, OR, NOT, Boolean SATisfiability
- Do you have basic understanding of Compilers?
  - Control Flow Graphs, Operational Semantics, Intermediate Representation
- Have you used a SAT / Theorem Prover / Constriant Solver / SMT ?

My Background

Since 10/2013:

Assistant Prof. at uWaterloo, Canada

Post-Doc at U of Washington in Seattle, USA

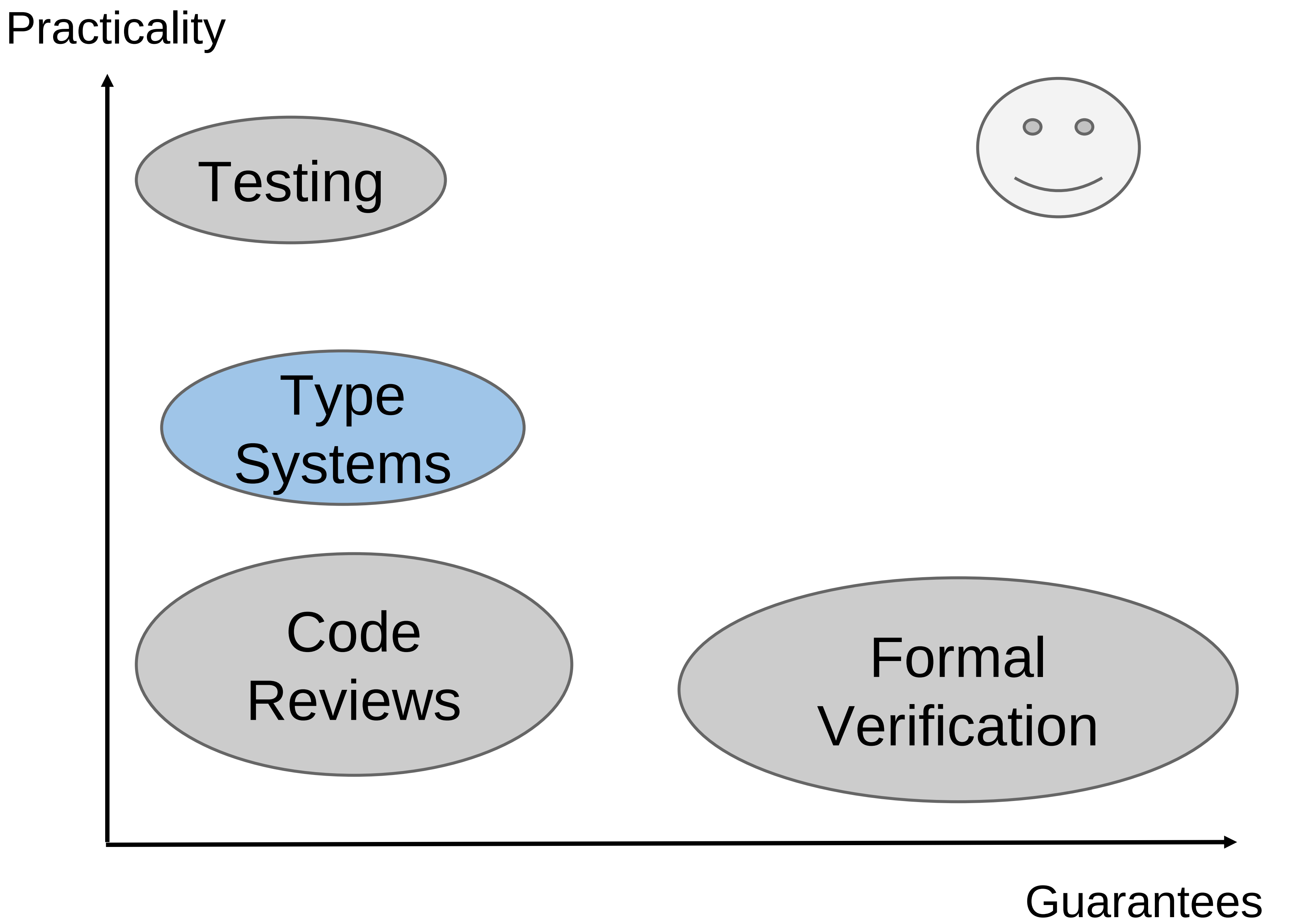
Dr. sc. at ETH Zurich, Switzerland

Dipl.-Ing. from Salzburg, Austria

Worked at a Startup in California, USA

MSc from somewhere in Ohio, USA

Born in Austria


Practicality
Testing
Type Systems
Code Reviews
Formal Verification
Guarantees

Java's type system is too weak

Type checking prevents many errors

```
int i = “hello”; // error
```

Type checking doesn't prevent enough errors

```
System.console().readLine();
Collections.emptyList().add(“one”);
dbStatement.executeQuery(userData);
```

Static types: not expressive enough

Null pointer exceptions

```
String op(Data in) {
    return “transform: ” + in.getF();
}
...
String s = op(null);
```

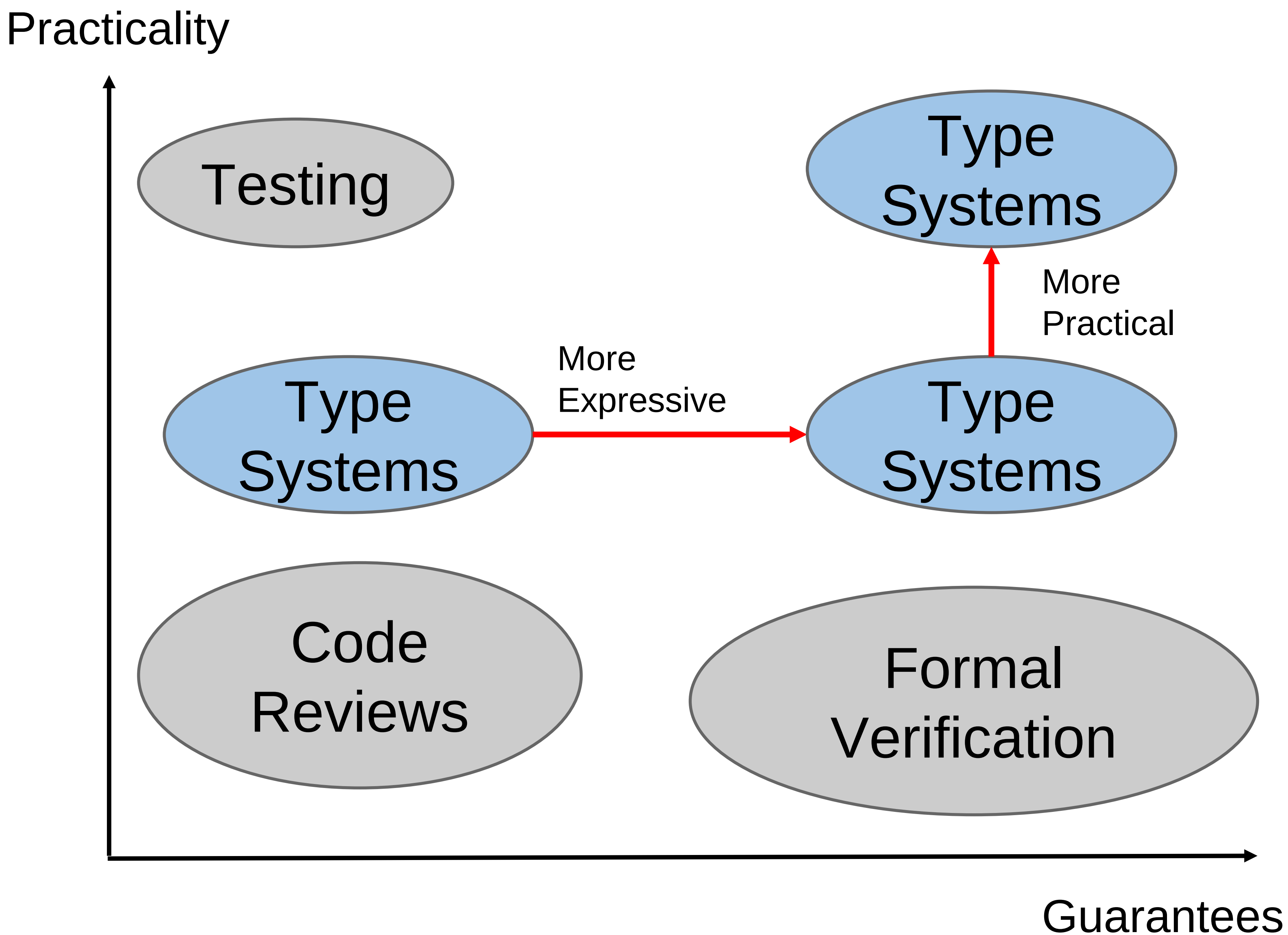
Many other properties can't be expressed

Prevent null pointer exceptions

Type system that statically guarantees that the program only dereferences known non-null references

Types of data

`@NonNull`	reference is never null
`@Nullable`	reference may be null


Practicality
Testing
Type Systems
More Practical
Type Systems
More Expressive
Type Systems
Code Reviews
Formal Verification
Guarantees

Java 8 extends annotation syntax

Annotations on all occurrences of types:

```
@Untainted String query;
List<@NonNull String> strings;
myGraph = (@Immutable Graph) tmp;
class UnmodifiableList<T>
  implements @Readonly List<T> {}
```

Stored in classfile

Handled by javac, javap, javadoc, …

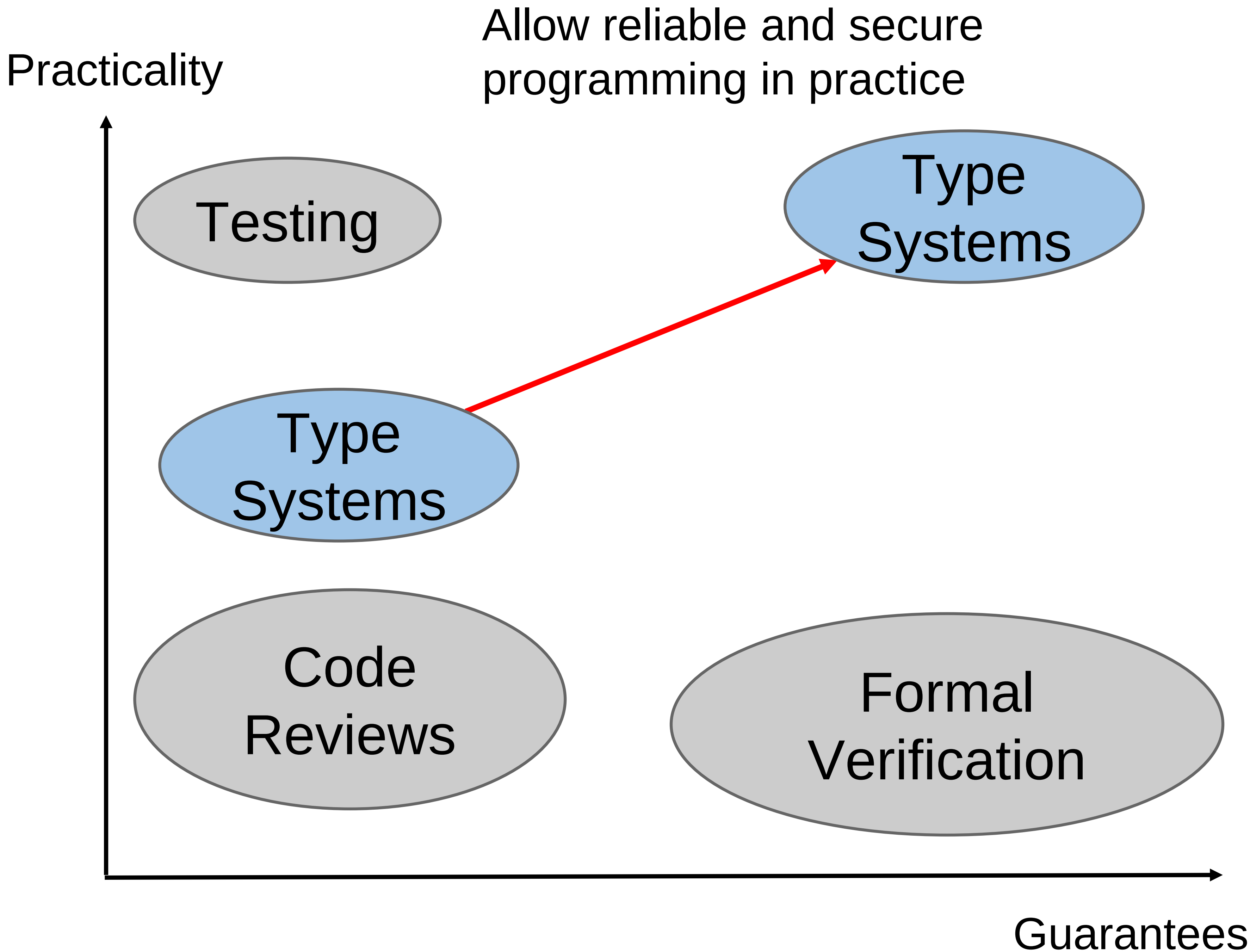
The Checker Framework

A framework for pluggable type checkers

“Plugs” into the OpenJDK compiler

```
javac -processor MyChecker …
```

Eclipse plug-in, Ant and Maven integration


Allow reliable and secure programming in practice
Practicality
Testing
Type Systems
Type Systems
Code Reviews
Formal Verification
Guarantees

Formalizations

$$\begin{array}{lclcl}
\mathtt{h} & \in & \mathtt{Heap} & = & \mathtt{Addr} \to \mathtt{Obj} \\
\iota & \in & \mathtt{Addr} & = & \text{Set of Addresses} \cup \{\mathtt{null}_a\} \\
\mathtt{o} & \in & \mathtt{Obj} & = & {}^{\mathtt{r}}\mathtt{Type},\ \mathtt{Fields} \\
{}^{\mathtt{r}}\mathtt{T} & \in & {}^{\mathtt{r}}\mathtt{Type} & = & \mathtt{OwnerAddr}\ \mathtt{ClassId}\mathtt{<}\overline{{}^{\mathtt{r}}\mathtt{Type}}\mathtt{>} \\
\mathtt{Fs} & \in & \mathtt{Fields} & = & \mathtt{FieldId} \to \mathtt{Addr} \\
\iota & \in & \mathtt{OwnerAddr} & = & \mathtt{Addr} \cup \{\mathtt{any}_a\} \\
{}^{\mathtt{r}}\Gamma & \in & {}^{\mathtt{r}}\mathtt{Env} & = & \overline{\mathtt{TVarId}\ {}^{\mathtt{r}}\mathtt{Type}};\ \overline{\mathtt{ParId}\ \mathtt{Addr}}
\end{array}$$

$$\begin{array}{lclcl}
\mathtt{P} & \in & \mathtt{Program} & ::= & \overline{\mathtt{Class}},\ \mathtt{ClassId},\ \mathtt{Expr} \\
\mathtt{Cls} & \in & \mathtt{Class} & ::= & \mathtt{class}\ \mathtt{ClassId}\mathtt{<}\overline{\mathtt{TVarI}} \\
 & & & & \mathtt{extends}\ \mathtt{ClassId}\mathtt{<}{}^{\mathtt{s}}\mathtt{Ty} \\
 & & & & \{\ \overline{\mathtt{FieldId}\ {}^{\mathtt{s}}\mathtt{Type};}\ \mathtt{Met} \\
{}^{\mathtt{s}}\mathtt{T} & \in & {}^{\mathtt{s}}\mathtt{Type} & ::= & {}^{\mathtt{s}}\mathtt{NType}\ \mid\ \mathtt{TVarId} \\
{}^{\mathtt{s}}\mathtt{N} & \in & {}^{\mathtt{s}}\mathtt{NType} & ::= & \mathtt{OM}\ \mathtt{ClassId}\mathtt{<}\overline{{}^{\mathtt{s}}\mathtt{Type}}\mathtt{>} \\
\mathtt{u} & \in & \mathtt{OM} & ::= & \\
\mathtt{mt} & \in & \mathtt{Meth} & ::= & \\
 & & \mathtt{MethSig} & ::= & \\
\mathtt{w} & \in & \mathtt{Purity} & ::= & \\
\mathtt{e} & \in & \mathtt{Expr} & ::= & \\
 & & & & \mathtt{Expr.MethId}\mathtt{<}{}^{\mathtt{s}}\mathtt{Type}\mathtt{>}(\mathtt{Expr})\ \mid \\
 & & & & \mathtt{new}\ {}^{\mathtt{s}}\mathtt{Type}\ \mid\ ({}^{\mathtt{s}}\mathtt{Type})\ \mathtt{Expr} \\
{}^{\mathtt{s}}\Gamma & \in & {}^{\mathtt{s}}\mathtt{Env} & ::= & \overline{\mathtt{TVarId}\ {}^{\mathtt{s}}\mathtt{NType}};\ \overline{\mathtt{ParId}\ {}^{\mathtt{s}}\mathtt{Type}}
\end{array}$$

$$\text{OS-Read}\ \frac{\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}', \iota_0 \qquad \iota_0 \neq \mathtt{null}_a \qquad \iota = \mathtt{h}'(\iota_0)\downarrow_2(\mathtt{f})}{\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0.\mathtt{f} \rightsquigarrow \mathtt{h}', \iota}$$

$$\text{OS-Upd}\ \frac{\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}_0, \iota_0 \qquad \iota_0 \neq \mathtt{null}_a \qquad \mathtt{h}_0, {}^{\mathtt{r}}\Gamma, \mathtt{e}_2 \rightsquigarrow \mathtt{h}_2, \iota \qquad \mathtt{h}' = \mathtt{h}_2[\iota_0.\mathtt{f} := \iota]}{\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 \rightsquigarrow \mathtt{h}',}$$

$$\text{GT-Upd}\ \frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \mathtt{u}_0\ \mathtt{C}_0\mathtt{<\_>} \qquad \mathtt{T}_1 = fType(\mathtt{C}_0, \mathtt{f}) \qquad \Gamma \vdash \mathtt{e}_2 : \mathtt{N}_0 \triangleright \mathtt{T}_1 \qquad \mathtt{u}_0 \neq \mathtt{any} \qquad rp(\mathtt{u}_0, \mathtt{T}_1)}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 : \mathtt{N}_0 \triangleright \mathtt{T}_1}$$

$$\text{GT-Read}\ \frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \_\ (}{\Gamma \vdash \mathtt{e}_0.\mathtt{f} : \mathtt{N}_0 \triangleright fType(\mathtt{C}_0, \mathtt{f})}$$

$$\begin{array}{l}
\mathtt{h} \vdash {}^{\mathtt{r}}\Gamma : {}^{\mathtt{s}}\Gamma \\
\mathtt{h} \vdash \iota_1 : dyn({}^{\mathtt{s}}\mathtt{N}, \mathtt{h}, {}^{\mathtt{r}}\iota\ ) \\
\mathtt{h} \vdash \iota_2 : dyn({}^{\mathtt{s}}\mathtt{T}, \iota_1, \mathtt{h}(\iota_1)\downarrow_1) \\
{}^{\mathtt{s}}\mathtt{N} = \mathtt{u}_N\ \mathtt{C}_N\mathtt{<>} \\
\mathtt{u}_N = \mathtt{this}_u \Rightarrow {}^{\mathtt{r}}\Gamma(\mathtt{this}) \\
free({}^{\mathtt{s}}\mathtt{T}) \subseteq dom(\mathtt{C}_N)
\end{array} \Bigg\} \implies \mathtt{h} \vdash \iota_2 : dyn({}^{\mathtt{s}}\mathtt{N} \triangleright {}^{\mathtt{s}}\mathtt{T}, \mathtt{h}, {}^{\mathtt{r}}\Gamma)$$

$$\text{DYN}\ \frac{{}^{\mathtt{r}}\mathtt{T} = \iota'\ \mathtt{\_<>} \qquad \iota \vdash {}^{\mathtt{r}}\mathtt{T}\ {}^{\mathtt{r}}\mathtt{<:}\ \iota'\ \mathtt{C}\mathtt{<}\overline{{}^{\mathtt{r}}\mathtt{T}}\mathtt{>} \qquad \iota \vdash {}^{\mathtt{r}}\mathtt{T}\ {}^{\mathtt{r}}\mathtt{<:}\ \iota'\ \mathtt{C}\mathtt{<}\overline{{}^{\mathtt{r}}\mathtt{T}_a}\mathtt{>} \Rightarrow \iota \vdash \overline{{}^{\mathtt{r}}\mathtt{T}}\ {}^{\mathtt{r}}\mathtt{<:}\ \overline{{}^{\mathtt{r}}\mathtt{T}_a} \qquad \mathrm{dom}(\mathtt{C}) = \overline{\mathtt{X}} \qquad \mathrm{free}({}^{\mathtt{s}}\mathtt{T}) \subseteq \overline{\mathtt{X}} \circ \overline{\mathtt{X}'}}{dyn({}^{\mathtt{s}}\mathtt{T}, \iota, {}^{\mathtt{r}}\mathtt{T}, (\overline{\mathtt{X}'\ {}^{\mathtt{r}}\mathtt{T}'};\_)) = {}^{\mathtt{s}}\mathtt{T}[\iota'/\mathtt{this}, \iota'/\mathtt{peer}, \iota/\mathtt{rep}, \mathtt{any}_a/\mathtt{any}_u, \overline{{}^{\mathtt{r}}\mathtt{T}/\mathtt{X}}, \overline{{}^{\mathtt{r}}\mathtt{T}'/\mathtt{X}'}]}$$

UNIVERSITY OF WATERLOO

SPARTA: Static Program Analysis for Reliable Trusted Apps

Security type system for Android apps

Guarantees no leakage of private information

UNIVERSITY OF WATERLOO

Crowd-sourced verification

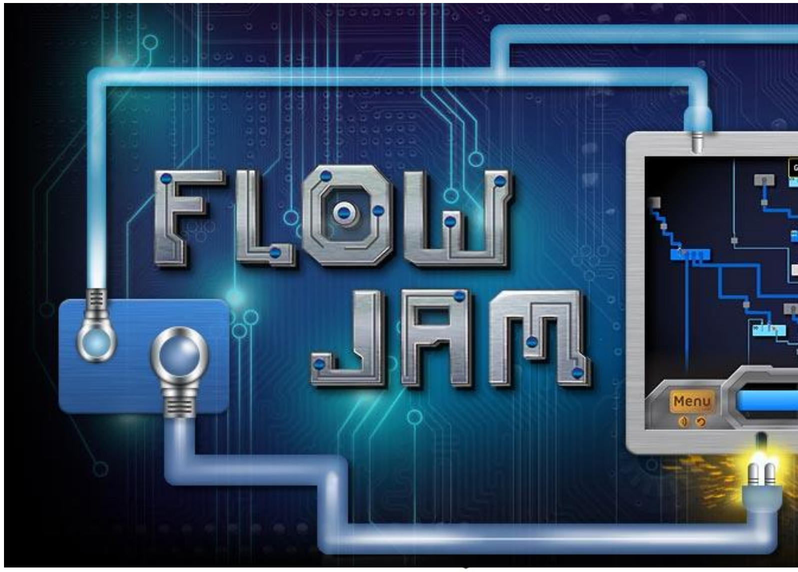
Make software verification easy and fun

Make the game accessible to everyone

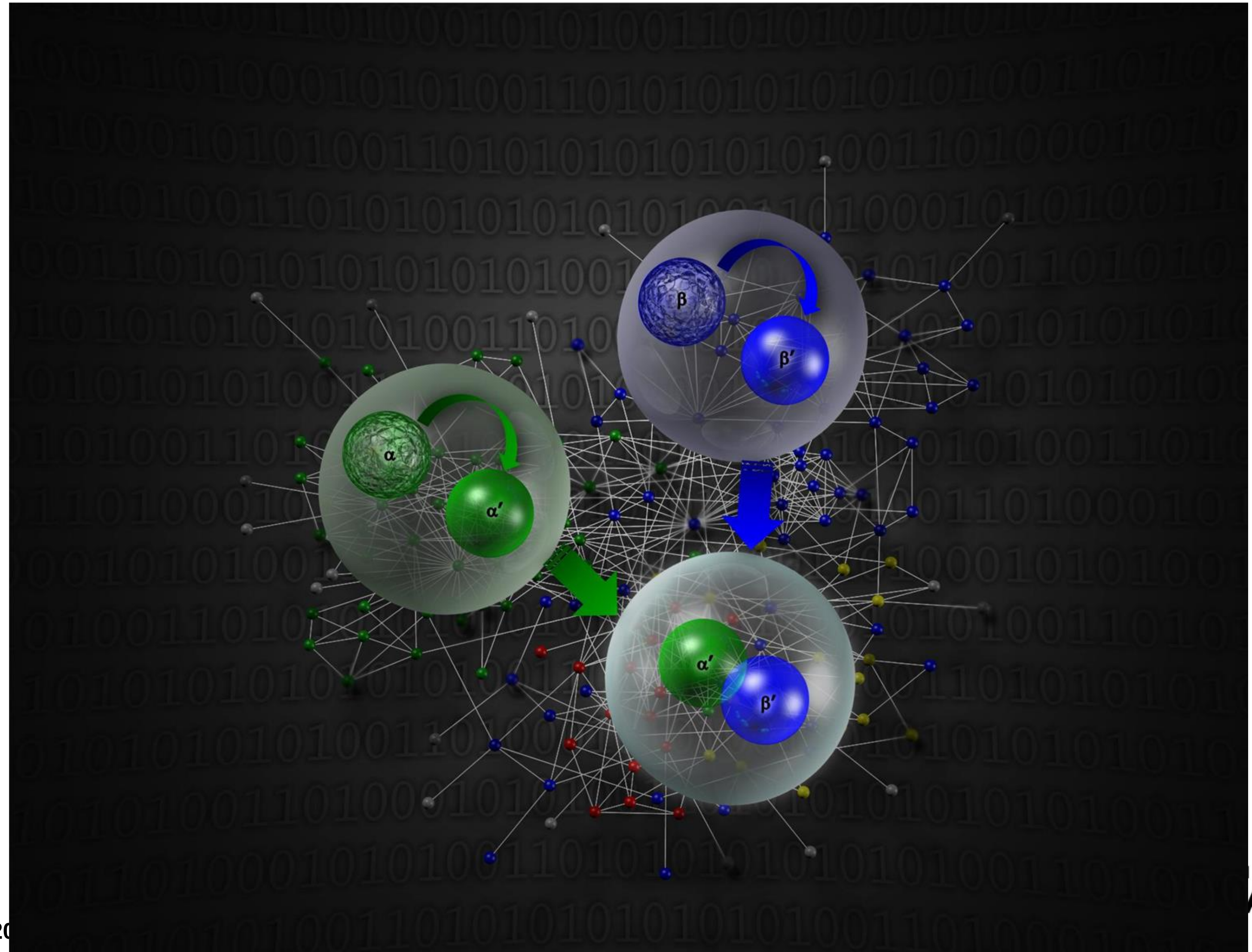
Harness the power of the crowd

Goal: Verify software while waiting

http://verigames.com/



MUSE: Mining and Understanding Software Enclaves



Software Testing, Quality Assurance, and Maintenance (STQAM)

ECE 653

Lecture 1: Tuesday, 07.05.2019

Introduction: Software Testing and Quality Assurance

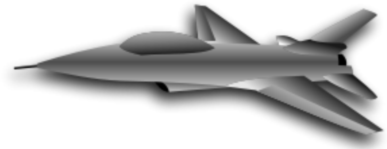
based on slides by Profs. Arie Gurfinkel and others

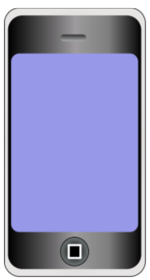
Werner Dietl

https://ece.uwaterloo.ca/~wdietl/

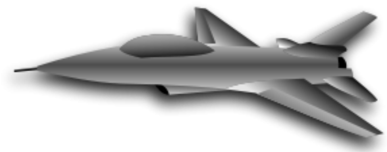
UNIVERSITY OF WATERLOO

Software is Everywhere





Software is Everywhere



"Software easily rates as the most poorly constructed, unreliable, and least maintainable technological artifacts invented by man"

Paul Strassman, former CIO of Xerox

Infamous Software Disasters

Between 1985 and 1987, **Therac-25** gave patients massive overdoses of radiation, approximately 100 times the intended dose. Three patients died as a direct consequence.

On February 25, 1991, during the Gulf War, an American **Patriot Missile** battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people.

On June 4, 1996 an unmanned **Ariane 5** rocket launched by the European Space Agency exploded forty seconds after lift-off. The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million.

http://www5.in.tum.de/~huckle/bugse.html

Proving that Android's, Java's and Python's sorting algorithm is broken (and showing how to fix it)

February 24, 2015 Envisage Written by Stijn de Gouw. $s

Tim Peters developed the Timsort hybrid sorting algorithm in 2002. It is a clever combination of ideas from merge sort and insertion sort, and designed to perform well on real world data. TimSort was first developed for Python, but later ported to Java (where it appears as java.util.Collections.sort and java.util.Arrays.sort) by Joshua Bloch (the designer of Java Collections who also pointed out that most binary search algorithms were broken). TimSort is today used as the default sorting algorithm for Android SDK, Sun's JDK and OpenJDK. Given the popularity of these platforms this means that the number of computers, cloud services and mobile phones that use TimSort for sorting is well into the billions.

http://envisage-project.eu/proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it/

Why so many bugs?

Software Engineering is very complex

- Complicated algorithms
- Many interconnected components
- Legacy systems
- Huge programming APIs
- …

Software Engineers need better tools to deal with this complexity!

UNIVERSITY OF WATERLOO

What Software Engineers Need Are …

Tools that give better confidence than *ad-hoc* testing while remaining easy to use

And at the same time, are

- … fully automatic
- … (reasonably) easy to use
- … provide (measurable) guarantees
- … come with guidelines and methodologies to apply effectively
- … apply to real software systems

UNIVERSITY OF WATERLOO

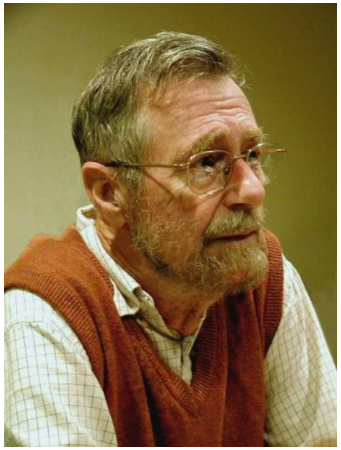
Testing

Software validation the “old-fashioned” way:

- Create a test suite (set of test cases)
- Run the test suite
- Fix the software if test suite fails
- Ship the software if test suite passes

"Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."

Edsger W. Dijkstra



Very hard to test the portion inside the "if" statement!

```
x = read();
if (hash(x) == 10) {
    ...
}
```

Hypothetical program

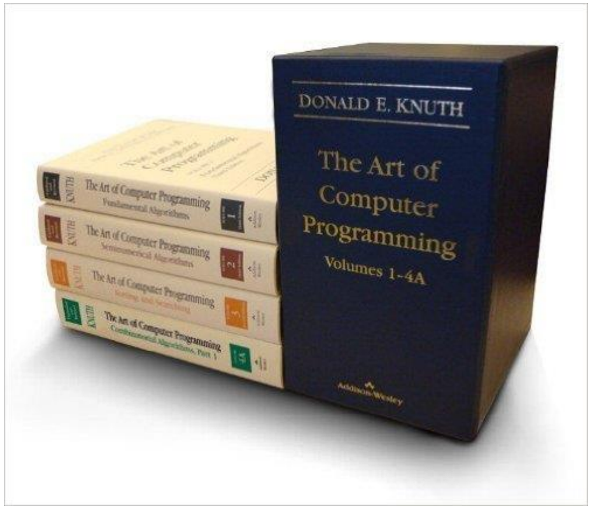
UNIVERSITY OF WATERLOO

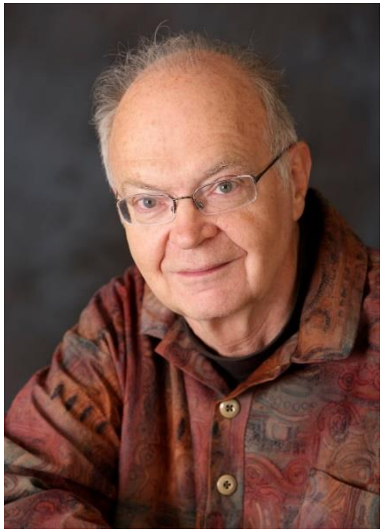
“Beware of bugs in the above code; I have only proved it correct, not tried it.”

Donald Knuth

You can only verify what you have specified.

Testing is still important, but can we make it less impromptu?





UNIVERSITY OF WATERLOO

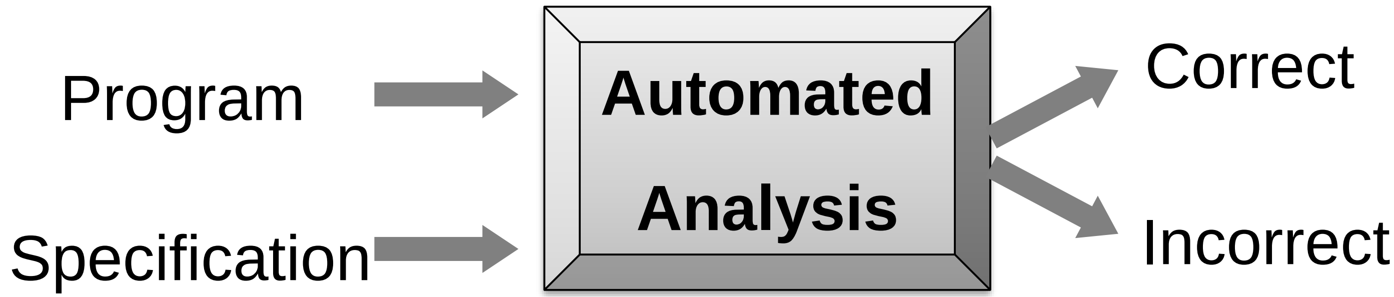
Verification / Quality Assurance

Verification: formally prove that a computing system satisfies its specifications

- **Rigor**: well established mathematical foundations
- **Exhaustiveness**: considers all possible behaviors of the system, i.e., finds all errors
- **Automation**: uses computers to build reliable computers

Formal Methods: general area of research related to program specification and verification

Ultimate Goal: Static Program Verification



Reasoning statically about behavior of a program without executing it

- compile-time analysis
- exhaustive, considers all possible executions under all possible environments and inputs

The *algorithmic* discovery of *properties* of program by *inspection* of the *source text*

Manna and Pnueli

Also known as *static analysis*, *program verification*, *formal methods*, etc.

UNIVERSITY OF WATERLOO

Turing, 1936: “undecidable”

Undecidability

A problem is undecidable if there does not exist a Turing machine that can solve it

- i.e., not solvable by a computer program

The halting problem

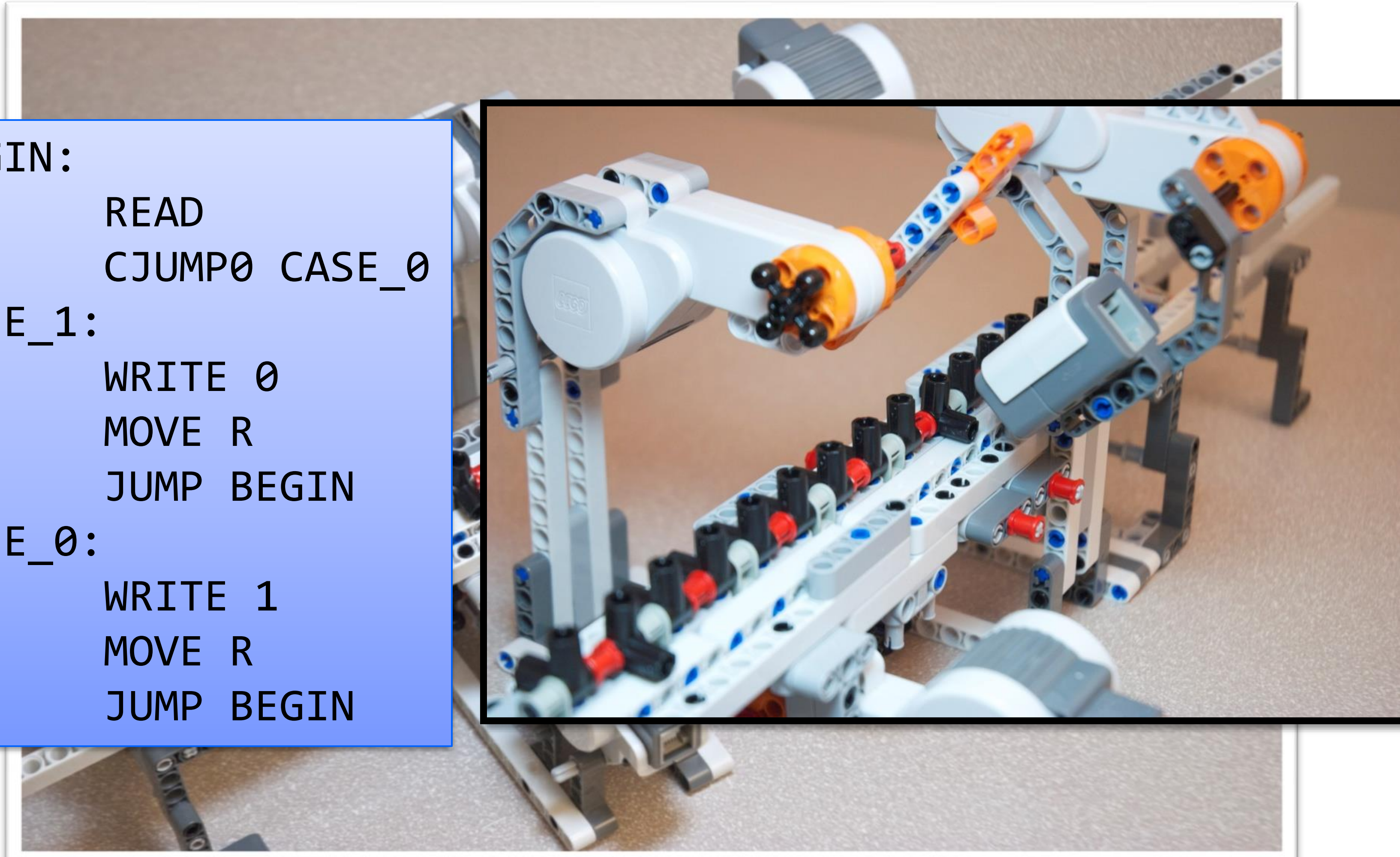
- does a program P terminate on input I
- proved undecidable by Alan Turing in 1936
- https://en.wikipedia.org/wiki/Halting_problem

Rice's Theorem

- for any non-trivial property of partial functions, no general and effective method can decide whether an algorithm computes a partial function with that property
- in practice, this means that there is no machine that can always decide whether the language of a given Turing machine has a particular nontrivial property
- https://en.wikipedia.org/wiki/Rice%27s_theorem

LEGO Turing Machine

```
BEGIN:
        READ
        CJUMP0 CASE_0
CASE_1:
        WRITE 0
        MOVE R
        JUMP BEGIN
CASE_0:
        WRITE 1
        MOVE R
        JUMP BEGIN
```



by Soonho Kong. See http://www.cs.cmu.edu/~soonhok for building instructions.

Living with Undecidability

"Algorithms" that occasionally diverge

Limit programs that can be analyzed

- finite-state, loop-free

Partial (unsound) verification

Testing **Sym Exec**

- analyze only some executions up-to a fixed number of steps

Incomplete verification / Abstraction

Automated Verification

- analyze a superset of program executions

Programmer Assistance

- annotations, pre-, post-conditions, inductive invariants

Deductive Verification

Formal Software Analysis

J. McCarthy, “*A basis for mathematical theory of computation*”, 1963.

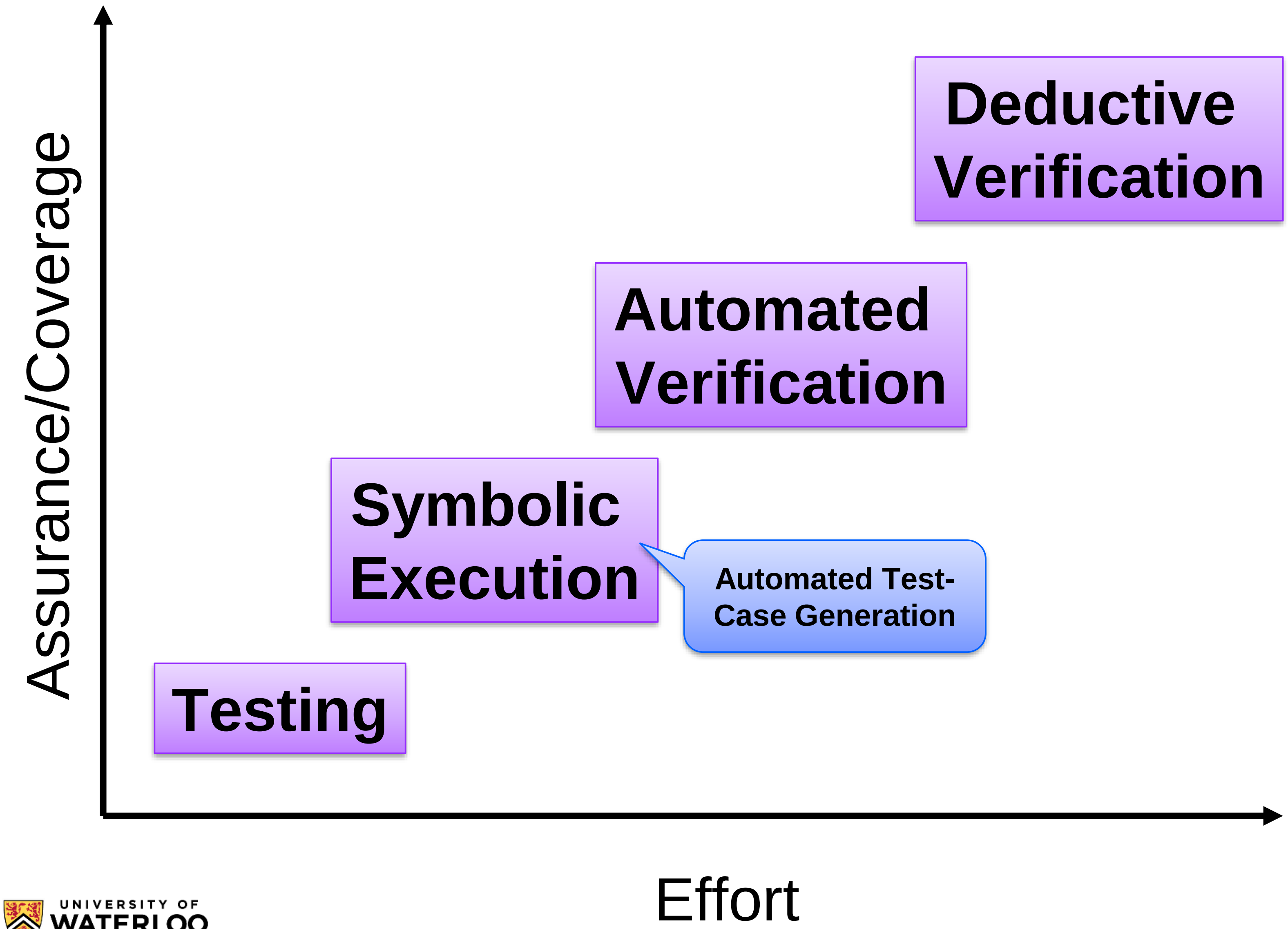
P. Naur, “*Proof of algorithms by general snapshots*”, 1966.

R. W. Floyd, “*Assigning meaning to programs*”, 1967.

C.A.R Hoare, “*An axiomatic basis for computer programming*”, 1969.

E. W. Dijkstra: “*Guarded Commands, Nondeterminacy and Formal derivation* ”, 1975.

(User) Effort vs (Verification) Assurance



UNIVERSITY OF WATERLOO

Why are Testing and Verification Necessary

Why Test?

Why Verify?

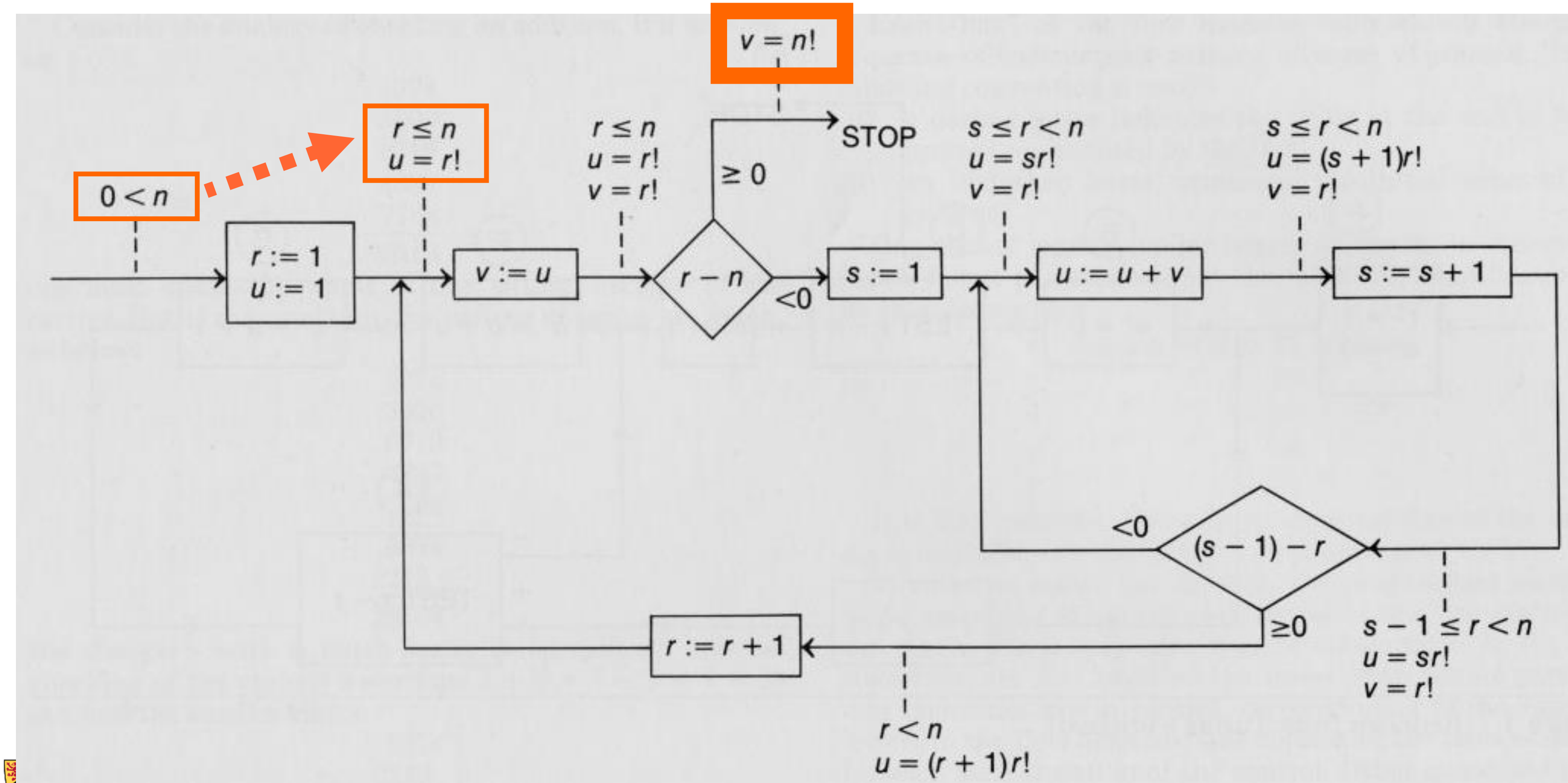
What is Verification? How is it different from Testing?

Turing, 1949

Alan M. Turing. “Checking a large routine”, 1949

How can one check a routine in the sense of making sure that it is right?

programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.



```
method factorial_turing (n: int) returns (v: int)
{
  var r := 1;
  var u := 1;

  while (true)
  {
    v := u;
    if (r - n ≥ 0)
    { return v; }
    var s := 1;
    while (true)
    {
      u := u + v;
      s := s + 1;
      if ((s - (r + 1)) ≥ 0)
      {break;}
    }
    r := r + 1;
  }
}
```

UNIVERSITY OF WATERLOO

```
method factorial (n: int) returns (v:int)


{
  v := 1;
  if (n == 1) { return v; }
  var i := 2;
  while (i <= n)


  {
    v := i * v;
    i := i + 1;
  }
  return v;
}
```

```
method factorial (n: int) returns (v:int)
  requires n >= 0;
  ensures v = fact(n);
{
  v := 1;
  if (n <= 1) { return v; }
  var i := 2;
  while (i <= n)
    invariant i <= n + 1
    invariant v = fact(i - 1)
  {
    v := i * v;
    i := i + 1;
  }
  return v;
}
```

Specification

Inductive Invariant

Proving inductive invariants

The main step is to show that the invariant is preserved by one execution of the loop

```
assume(i <= n + 1);
assume(v == fact(i - 1));
assume(i <= n);
v := i * v;
i := i + 1;
assert(i <= n + 1);
assert(v == fact(i - 1));
```

Correctness of a loop-free program can (often) be decided by a Theorem Prover or a Satisfiability Modulo Theory (SMT) solver.

Proving inductive invariants

The main step is to show that the invariant is preserved by one execution of the loop

```
(i0 <= n0+1)      &&
(v0 == (i0-1)!)   &&
(i0 <= n0)        &&
(v1 = i0 * v0)    &&
(i1 = i0 + 1)
➔
((i1 <= n0+1)     &&
(v1 == (i1-1)!))
```

```
assume(i <= n+1);
assume(v == fact(i-1));
assume(i <= n);
v := i*v;
i := i+1;

assert(i<=n+1);
assert(v == fact(i-1));
```

Correctness of a loop-free program can (often) be decided by a Theorem Prover or a Satisfiability Modulo Theory (SMT) solver.

Automated Verification

Deductive Verification

- A user provides a program and a verification certificate
  - e.g., inductive invariant, pre- and post-conditions, function summaries, etc.
- A tool automatically checks validity of the certificate
  - this is not easy! (might even be undecidable)
- Verification is manual but machine certified

Algorithmic Verification

- A user provides a program and a desired specification
  - e.g., program never writes outside of allocated memory
- A tool automatically checks validity of the specification
  - and generates a verification certificate if the program is correct
  - and generates a counterexample if the program is not correct
- Verification is completely automatic – “push-button”

Available Tools

Testing

- many tools actively used in industry. We will use Python **unittest**

Symbolic Execution / Automated Test-Case Generation

- mostly academic tools with emerging industrial applications
- KLEE, S2E, jDART, Pex (now Microsoft IntelliTest)

Automated Verification

- built into compilers, many lightweight static analyzers
  - clang analyzer, Facebook Infer, Coverity, …
- academic pushing the coverage/automation boundary
  - SeaHorn (my tool), JayHorn, CPAChecker, SMACK, T2, …

(Automated) Deductive Verification

- academic, still rather hard to use, we'll experience in class ☺
- Dafny/Boogie (Microsoft), Viper, Why3, KeY, ...

Key Challenges

Testing

- Coverage

Symbolic Execution and Automated Verification

- Scalability

Deductive Verification

- Usability

Common Challenge

- Specification / Oracle

Calendar Description

Software Testing, Quality Assurance and Maintenance

Introduces students to systematic testing of software systems. Software verification, reviews, metrics, quality assurance, and prediction of software reliability and availability. Related management issues.

Topics Covered in the Course

Foundations

- syntax, semantics, abstract syntax trees, visitors, control flow graphs

Testing

- coverage: structural, dataflow, and logic

Symbolic Execution / Automated Test-Case Generation

- using SMT solvers, constraints, path conditions, exploration strategies
- building a (toy) symbolic execution engine

Deductive Verification

- Hoare Logic, weakest pre-condition calculus, verification condition generation
- verifying algorithm using Dafny, building a small verification engine

Automated Verification

- (basics of) software model checking

Frequently Asked Questions

Is this course practical?

Is this course easy / hard?

What knowledge from the course is applicable to a developer?

Is it a compilers course?

Is it a logic course?

Do I have to attend the lectures?

What are most useful skills learned in the course?

- Foundations of testing and verification
- State-of-the-art tools and technique to automate testing and reasoning
- Understanding the difference between wishful thinking (I hope it works) and a strong argument (I know it works, here is why…)

Software Testing, Quality Assurance, and Maintenance (STQAM)

ECE 653

Lecture 1: Tuesday, 07.05.2019

Fault, Error, and Failure

based on slides by Profs. Arie Gurfinkel, Lin Tan, and others

Werner Dietl

https://ece.uwaterloo.ca/~wdietl/

UNIVERSITY OF WATERLOO

Terminology, IEEE 610.12-1990

Fault -- often referred to as Bug [Avizienis'00]

- A static defect in software (incorrect lines of code)

Error

- An incorrect internal state (unobserved)

Failure

- External, incorrect behaviour with respect to the expected behaviour (observed)

Not used consistently in literature!

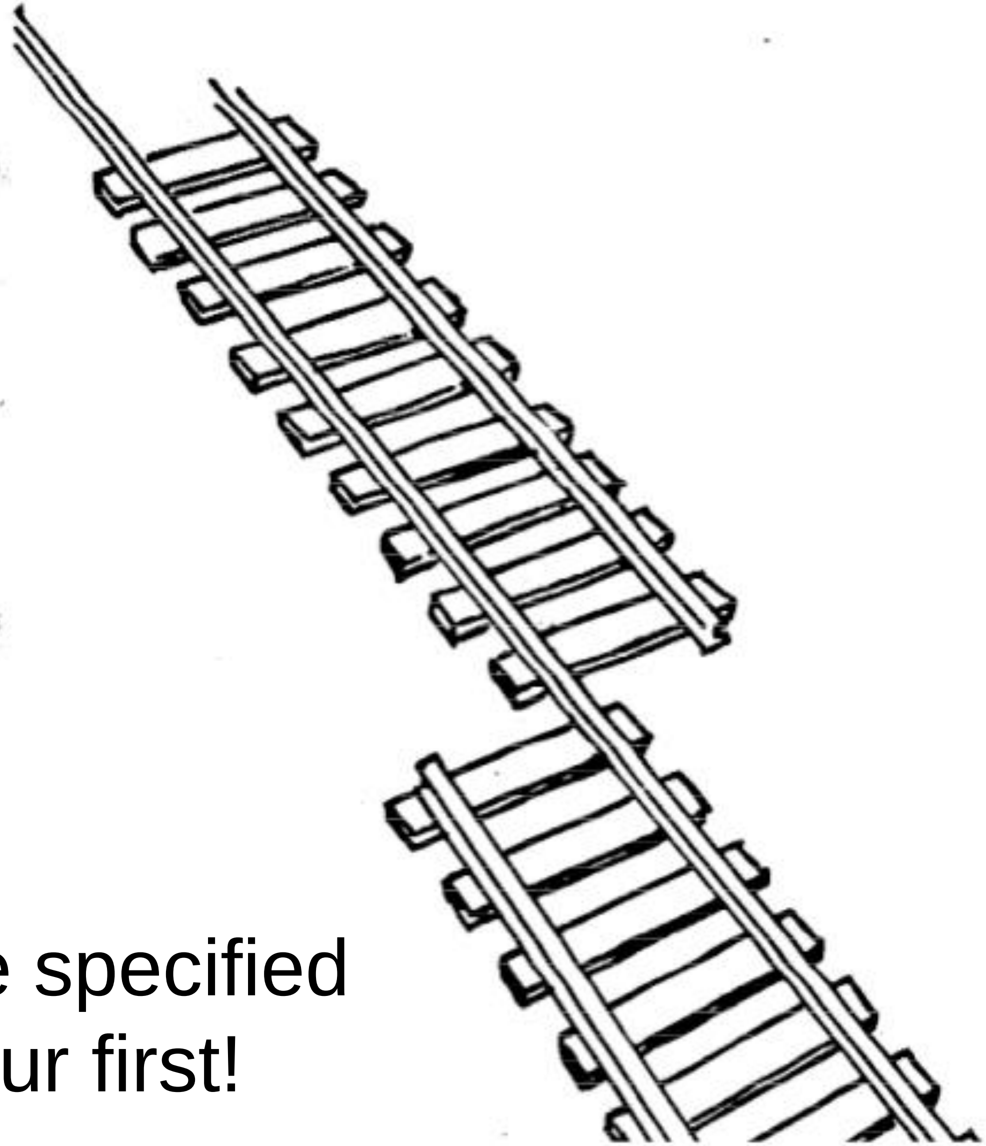
What is this?

A fault?

An error?

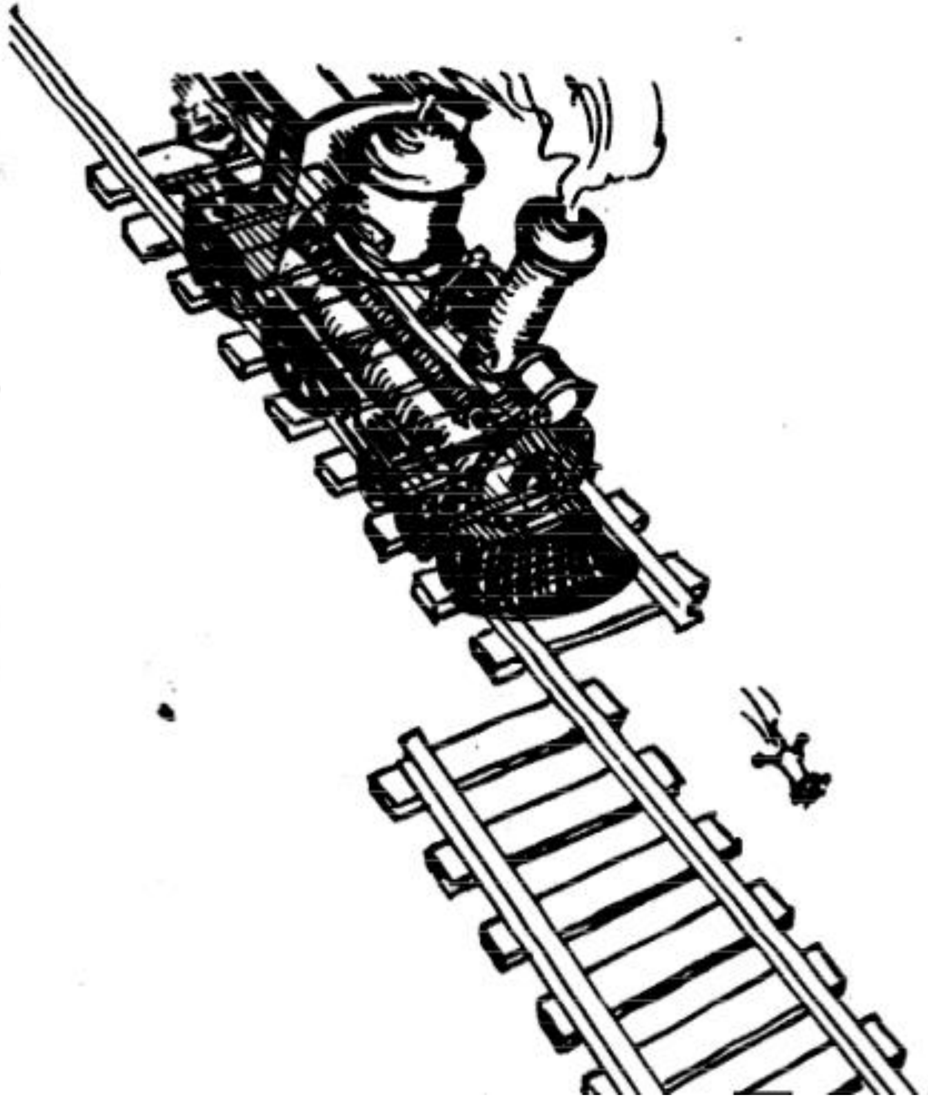
A failure?

We need to describe specified and desired behaviour first!



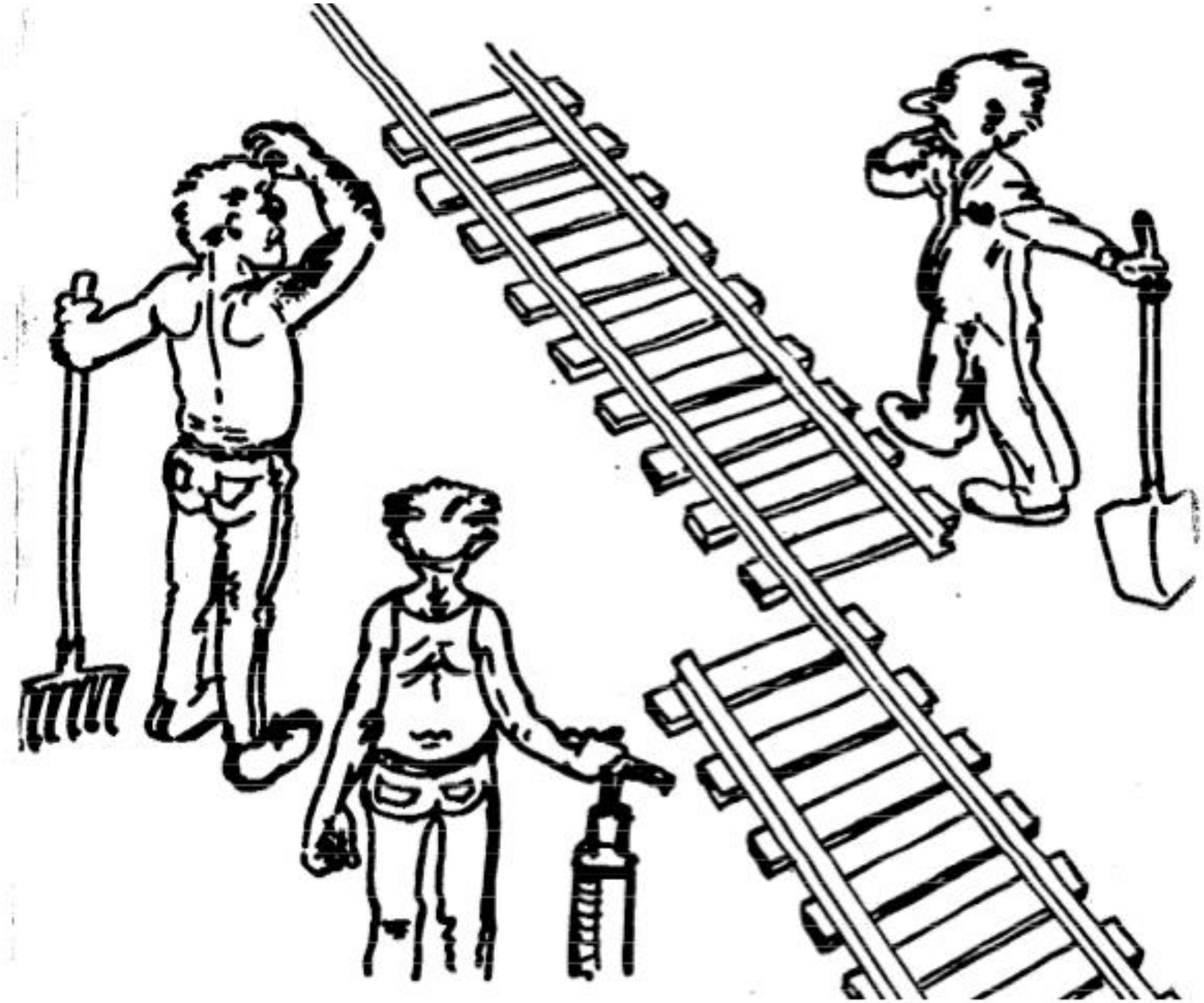
Bernd Bruegge & Allen H. Dutoit. Object-Oriented Software Engineering: Using UML, Patters, and Java

Erroneous State ("Error")



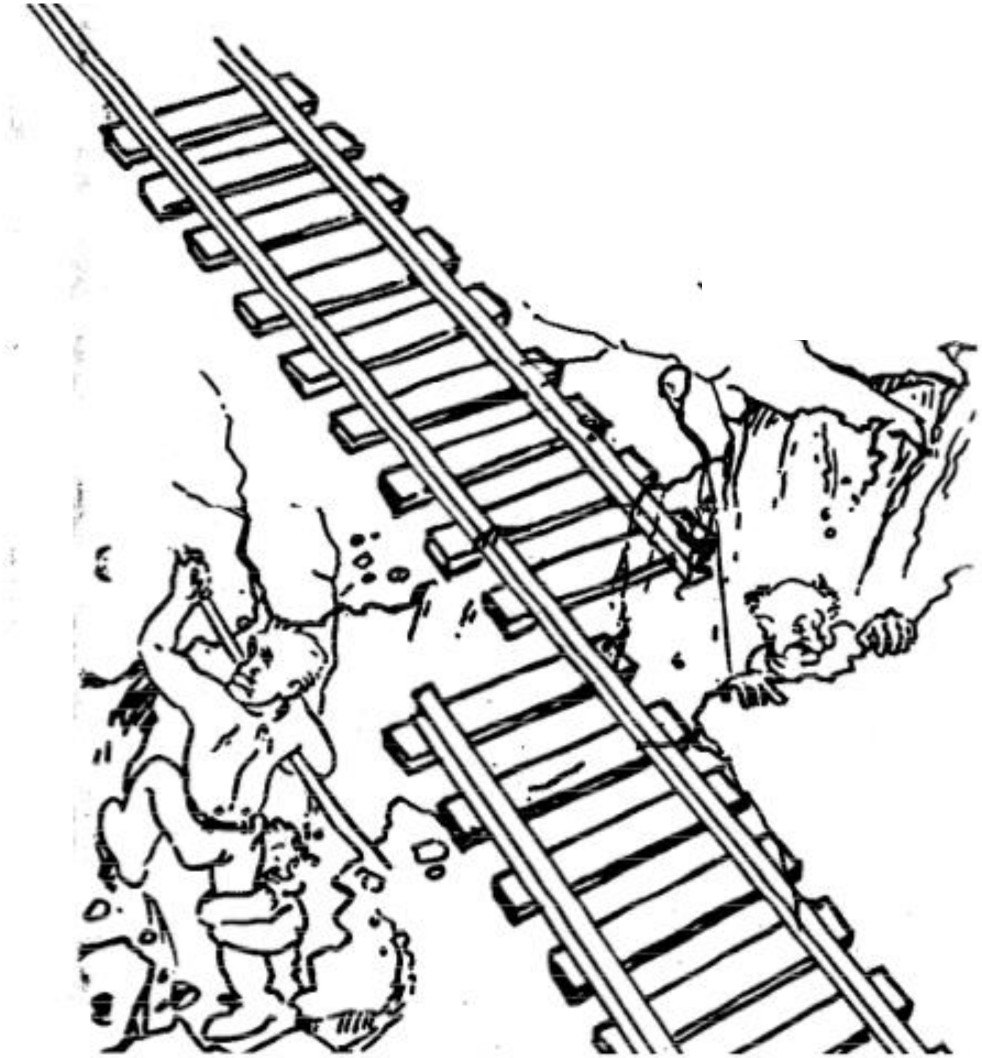
Bernd Bruegge & Allen H. Dutoit. Object-Oriented Software Engineering: Using UML, Patters, and Java

Design Fault



Bernd Bruegge & Allen H. Dutoit. Object-Oriented Software Engineering: Using UML, Patters, and Java

Mechanical Fault



Example: Fault, Error, Failure

```
public static int numZero (int[] x) {
//Effects: if x==null throw NullPointerException
//         else return the number of occurrences of 0 in x
   int count = 0;
   for (int i = 1; i <x.length; i++) {
      if (x[i]==0) {
         count++;
      }
   }
   return count;
```

Error State:
x = [2,7,0]
i =1
count =0
PC=first iteration **for**

Expected State:
x = [2,7,0]
i =0
count =0
PC=first iteration **for**

Fix: `for(int i=0; i<x.length; i++)`

x = [2,7,0], fault executed, error, no failure

x = [0,7,2], fault executed, error, failure

State of the program: x, i, count, PC

Exercise: The Program

```
/* Effect: if x==null throw NullPointerException.
   Otherwise, return the index of the last element
   in the array ‘x’ that equals integer ’y’.
   Return -1 if no such element exists.        */

public int findLast (int[] x, int y) {
for (int i=x.length-1; i>0; i--) {
     if (x[i] == y) { return i; }
  }
  return -1;
}

/* test 1: x=[2,3,5], y=2;
   expect: findLast(x,y) == 0
   test 2: x=[2,3,5,2], y=2;
   expect: findLast(x,y) == 3 */
```

Exercise: The Problem

Read this faulty program, which includes a test case that results in failure. Answer the following questions.

- (a) Identify the fault, and fix the fault.
- (b) If possible, identify a test case that does not execute the fault.
- (c) If possible, identify a test case that executes the fault, but does not result in an error state.
- (d) If possible identify a test case that results in an error, but not a failure. Hint: Don't forget about the program counter.
- (e) For the given test case ‘test1’, identify the first error state. Be sure to describe the complete state.

States

State 0:

- x = [2,3,5]
- y = 2
- i = undefined
- PC = findLast(...)

- State 1:
  - x = [2,3,5]
  - y = 2
  - i = undefined
  - PC = before i =x.length-1;

- State 2:
  - x = [2,3,5]
  - y = 2
  - i = 2
  - PC = after i =x.length-1;

- State 3:
  - x = [2,3,5]
  - y = 2
  - i = 2
  - PC = i>0;

States

- State 3:
  - x = [2,3,5]
  - y = 2
  - i = 2
  - PC = i>0;

- State 4:
  - x = [2,3,5]
  - y = 2
  - i = 2
  - PC = if (x[i] ==y);

- State 5:
  - x = [2,3,5]
  - y = 2
  - i = 1
  - PC = i--;

- State 6:
  - x = [2,3,5]
  - y = 2
  - i = 1
  - PC = i>0;

- State 7:
  - x = [2,3,5]
  - y = 2
  - i = 1
  - PC = if (x[i] ==y);

- State 8:
  - x = [2,3,5]
  - y = 2
  - i = 0
  - PC = i--;

UNIVERSITY OF WATERLOO

States

Incorrect Program

- State 8:
  - x = [2,3,5]
  - y = 2
  - i = 0
  - PC = i--;

- State 9:
  - x = [2,3,5]
  - y = 2
  - i = 0
  - PC = i>0;

- State 10:
  - x = [2,3,5]
  - y = 2
  - i = 0 (undefined)
  - PC = return -1;

Correct Program

- State 10:
  - x = [2,3,5]
  - y = 2
  - i = 0
  - PC = if (x[i]==y);

Exercise: Solutions (1/2)

(a) The for-loop should include the 0 index:

- for (int i=x.length-1; i >= 0; i--)

(b) The null value for x will result in a NullPointerException before the loop test is evaluated, hence no execution of the fault.

- Input: x = null; y = 3
- Expected Output: NullPointerException
- Actual Output: NullPointerException

(c) For any input where y appears in a position that is not position 0, there is no error. Also, if x is empty, there is no error.

- Input: x = [2, 3, 5]; y = 3;
- Expected Output: 1
- Actual Output: 1

Exercise: Solutions (2/2)

(d) For an input where y is not in x, the missing path (i.e. an incorrect PC on the final loop that is not taken, normally i = 2, 1, 0, but this one has only i = 2, 1,) is an error, but there is no failure.

- Input: x = [2, 3, 5]; y = 7;
- Expected Output: -1
- Actual Output: -1

(e) Note that the key aspect of the error state is that the PC is outside the loop (following the false evaluation of the 0>0 test. In a correct program, the PC should be at the if-test, with index i==0.

- Input: x = [2, 3, 5]; y = 2;
- Expected Output: 0
- Actual Output: -1
- First Error State:
  - x = [2, 3, 5]
  - y = 2;
  - i = 0 (or undefined);
  - PC = return -1;

RIP Model

Three conditions must be present for an error to be observed (i.e., failure to happen):

- Reachability: the location or locations in the program that contain the fault must be reached.
- Infection: After executing the location, the state of the program must be incorrect.
- Propagation: The infected state must propagate to cause some output of the program to be incorrect.

HOW DO WE DEAL WITH FAULTS, ERRORS, AND FAILURES?

Addressing Faults at Different Stages

Fault Avoidance

Better Design, Better PL, ...

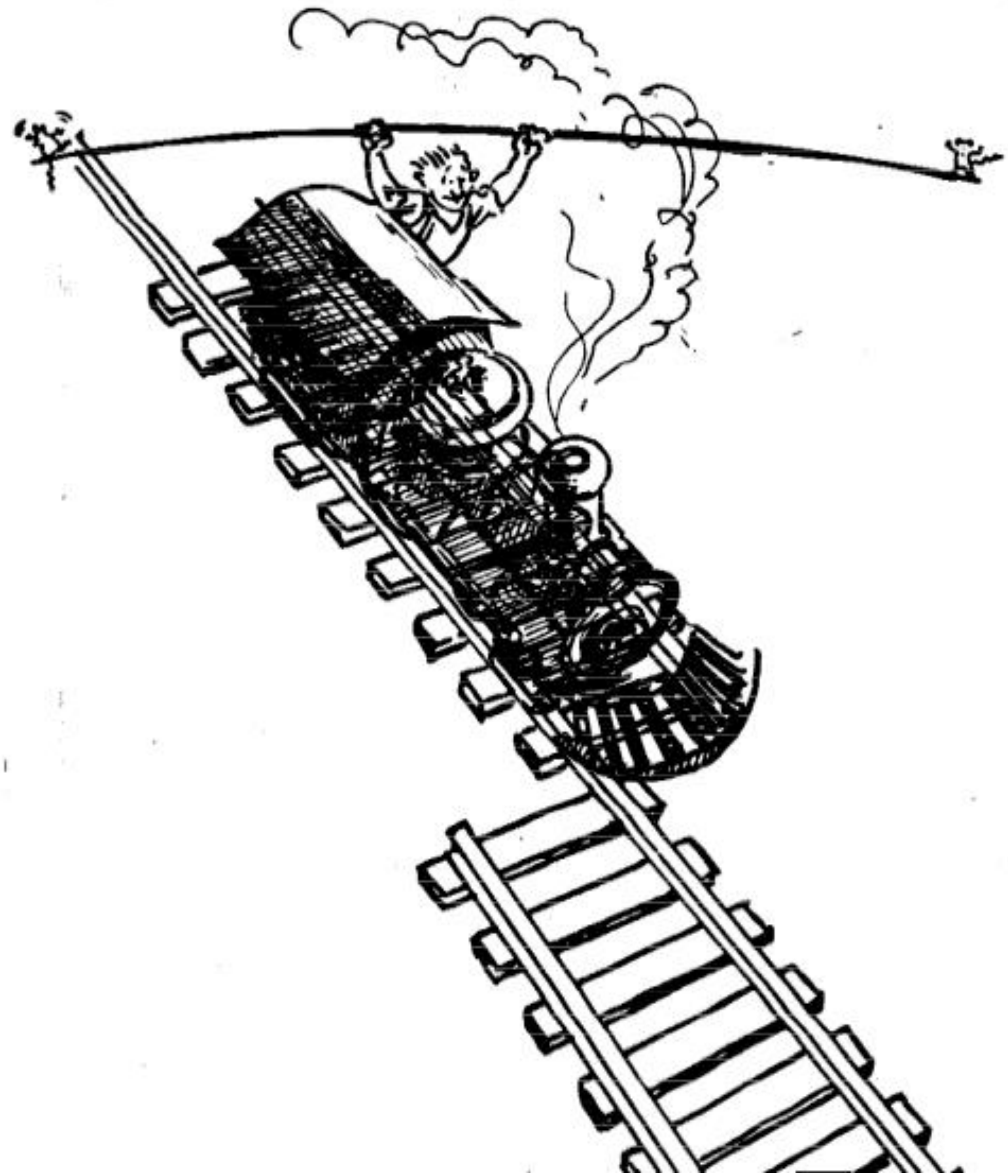
Fault Detection

Testing, Debugging, ...

Fault Tolerance

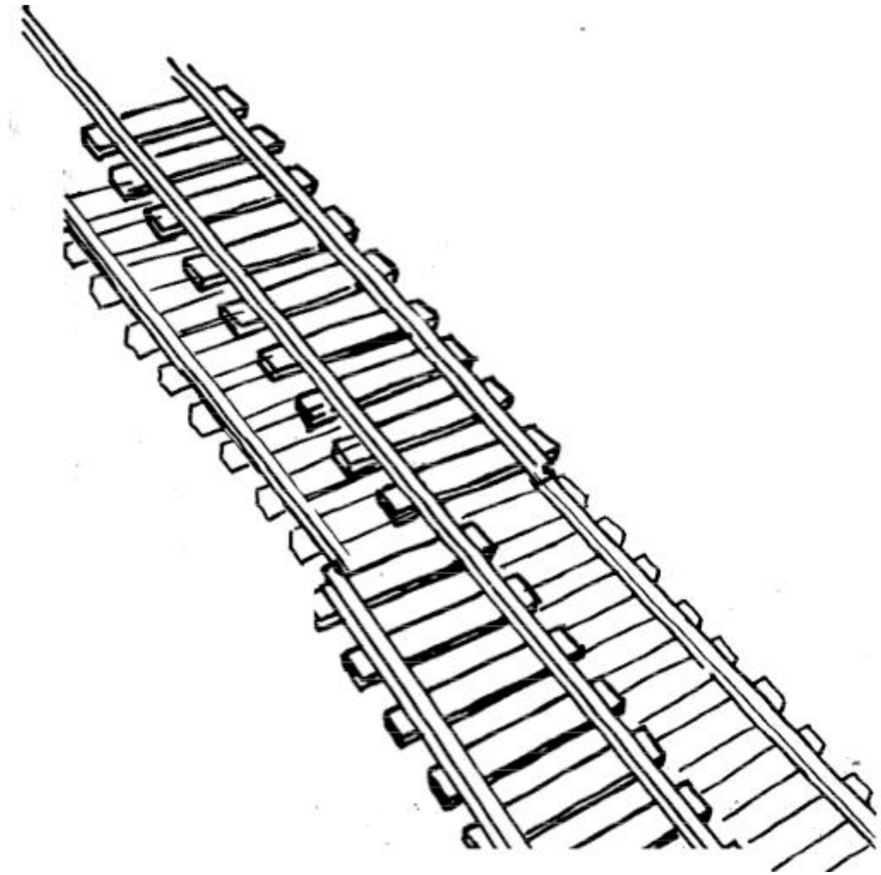
Redundancy, Isolation, ...

Declaring the Bug as a Feature

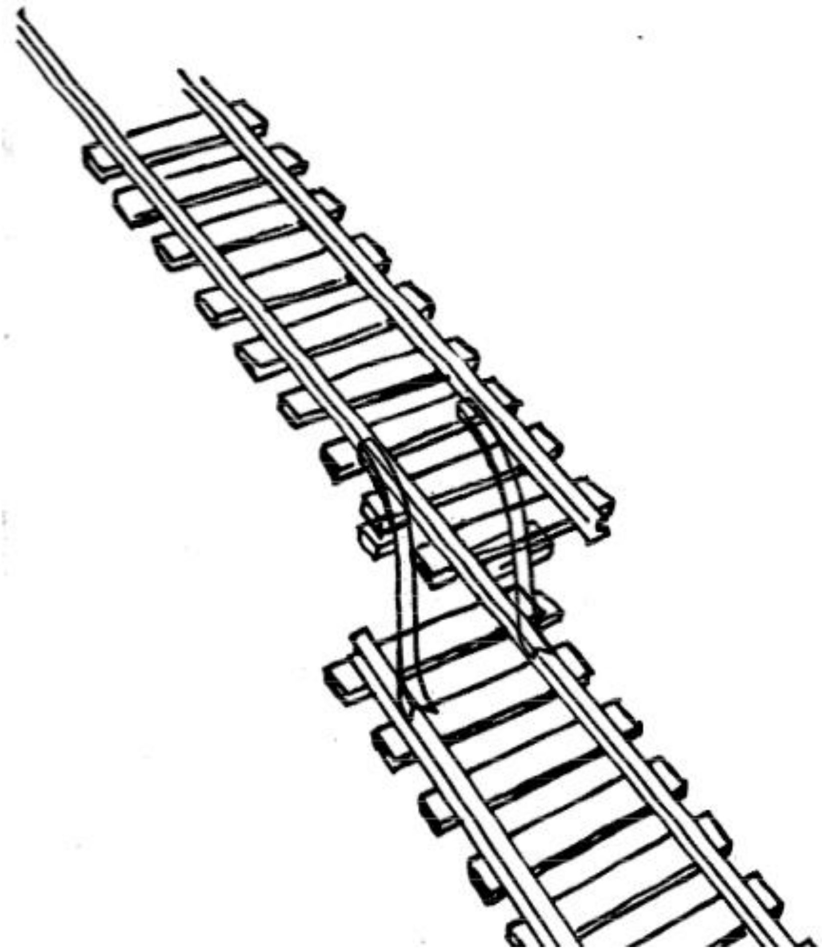


Bernd Bruegge & Allen H. Dutoit. Object-Oriented Software Engineering: Using UML, Patters, and Java

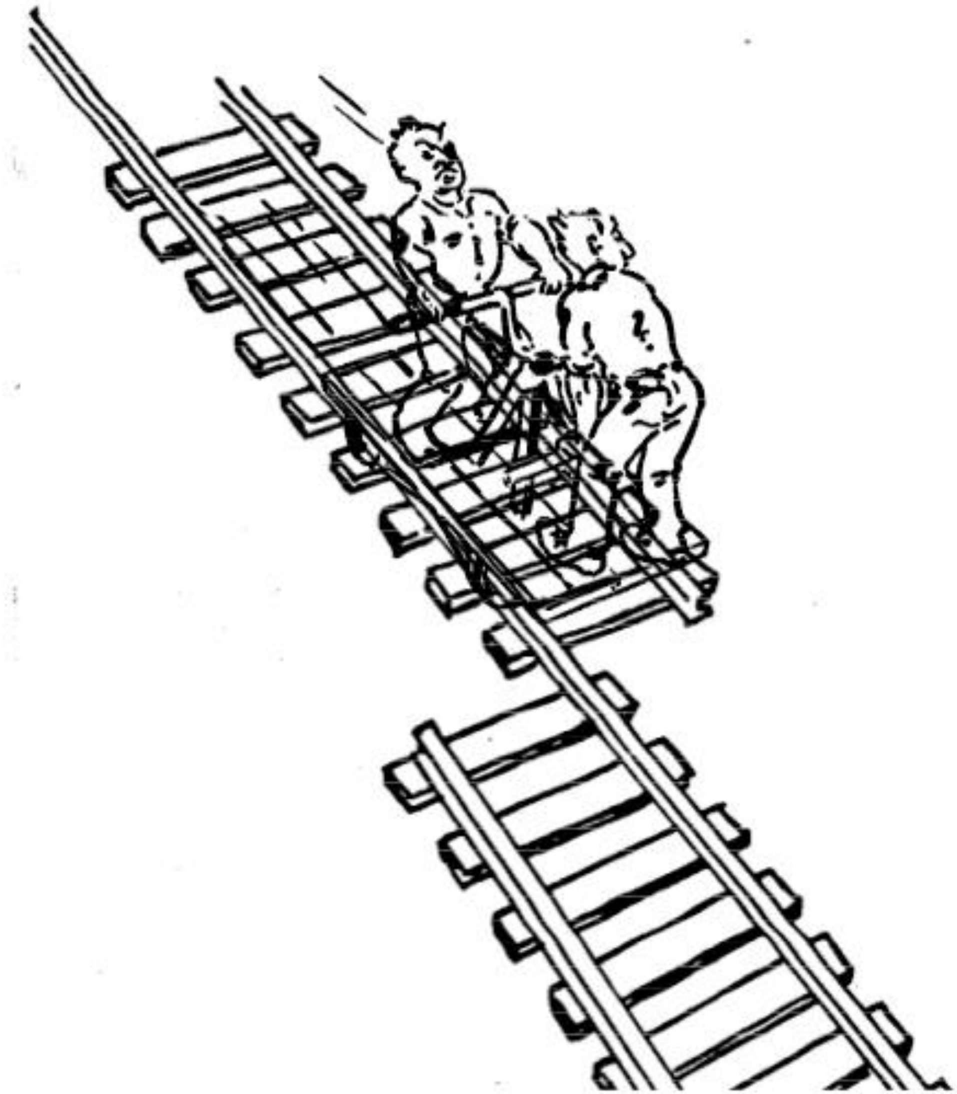
Modular Redundancy: Fault Tolerance



Patching: Fixing the Fault



Testing: Fault Detection



Testing vs. Debugging

Testing: Evaluating software by observing its execution

Debugging: The process of finding a fault given a failure

Testing is hard:

- Often, only specific inputs will trigger the fault into creating a failure.

Debugging is hard:

- Given a failure, it is often difficult to know the fault.

Testing is hard

```
if ( x - 100 <= 0 )
  if ( y - 100 <= 0 )
    if ( x + y - 200 == 0 )
      crash();
```

Only input `x=100 & y=100` triggers the crash

If x and y are 32-bit integers, what is the probability of a crash?

- $1 / 2^{64}$

Exercise: The Problem

```python
def pos_odd (x):
    """Ensures: returns the number of positive odd elements in the list x
       or throws an exception if x is not a list of numbers"""
    cnt = 0
    i = 0
    while i < len (x):
        if x[i] % 2 == 1:
            cnt = cnt + 1
        i = i + 1

    return cnt

# x = [-10,-9,0,99,100]
# r = pos_odd(x)
# assert (r == 1)
```

a) What is the fault in this program

b) Identify a test case that does not execute the fault

c) Identify a test case that results in an error but does not cause failure

d) Identify a test case that causes a failure but no error

e) For the test case `x = [-10, -9, 0, 99, 100]` the expected output is 1. Identify the first error state