

# Principles of Software Construction: Objects, Design, and Concurrency

## Functional Correctness – A Broader Perspective

*toad*

Spring 2014

**Christian Kästner**

Charlie Garrod

# Learning Goals

- Writing bug reports
- Apply Hoare-style verification to object-oriented programs
- Reason about inheritance with behavioral subtyping
- Apply static analysis tools
- Understand the tradeoffs among testing, formal verification and static analysis

# Bug Reports

# Reporting Defects

- Reproducible defects
  - Easier to find and fix
  - Easier to validate
  - Increased confidence
- Simple and general
  - More value doing the fix
- Non-antagonistic
  - State the problem
  - Don't blame

The screenshot displays the Eclipse Bugzilla interface for bug 141261. The header shows the Eclipse logo and 'Eclipse bugs' with version 3.20.3. The bug title is 'crash - Shell create, RepositionWindow() - Unexpected Eclipse crash RC3 (JavaNativeCrash)'. The last modified date is 2006-11-14 at 17:46:58. A bug list summary indicates 31 of 200 bugs, with links for navigation and search. The main form contains fields for Bug#, Product, Component, Status, Resolution, Assigned To, Hardware, OS, Version, Priority, Severity, Target, Milestone, Reporter, Add CC, and CC. The QA Contact, URL, Summary, Status, Whiteboard, and Keywords fields are also present. A table at the bottom shows attachment details with columns for Attachment, Type, Created, Size, and Actions. The bug 141261 depends on and blocks fields are empty. The votes section shows 0 votes with links to show or vote for the bug. The additional comments section is at the bottom.

**eclipse** Eclipse bugs  
bugzilla 3.20.3

**Bugzilla Bug 141261** crash - Shell create, RepositionWindow() - Unexpected Eclipse crash RC3 (JavaNativeCrash) Last modified:  
2006-11-14  
17:46:58

Bug List: (31 of 200) [First](#) [Last](#) [Prev](#) [Next](#) [Show last search results](#) [Search page](#) [Enter new bug](#)

[Eclipse] Bug#: 141261

Product:

Component:

Status:

Resolution:

Assigned To:

Hardware:

OS:

Version:

Priority:

Severity:

Target Milestone:

Reporter:

Add CC:

CC:

☐ Remove selected CCs

QA Contact:

URL:

Summary:

Status:

Whiteboard:

Keywords:

Attachment	Type	Created	Size	Actions
<a href="#">Create a New Attachment</a>	(proposed patch, testcase, etc.)			<a href="#">View All</a>

Bug 141261 depends on:

Bug 141261 blocks:

[Show dependency tree](#)

Votes: 0 [Show votes for this bug](#) [Vote for this bug](#)

Additional Comments:

# Social Issues in Defect Reporting

- There are differences between developer and tester culture
- Acknowledge that testers often deliver bad news
- Work hard to detect defects locally
  - Easier to narrow scope and responsibility
  - Less adversarial
- Don't measure performance in terms of defect reports

☐ [Reassign](#) bug to

☐ Reassign bug to default assignee and QA contact of selected component

[View Bug Activity](#) | [Format For Printing](#) | [Clone This Bug](#)

---

**Description:** [\[reply\]](#) Opened: 2005-07-25 07:03

I didn't even know that there was an undo feature inside the GUI editor, but today I accidentally pressed CTRL-Z instead of CTRL-S and the undo started... crashed.

Try adding some extension in the extensions page and then press CTRL-Z.

This is actually two bugs imho;

1. The details is very very poor so I really don't know what happened. A stacktrace would be great for debugging.
2. The undo obviously does not work correctly.

here is a screenshot of the crash:  
[http://mnemo.minimum.se/eclipse\\_crashes/eclipse\\_undo\\_crash.png](http://mnemo.minimum.se/eclipse_crashes/eclipse_undo_crash.png)

I don't have time for extensive reprod testing atm, maybe someone else can assist with this and see if they can get the plugin.xml editor to crash using weird combinations of editing and CTRL-Z undoing.

# Defect Tracking

- Always track defects and issues
- Issue: Bug, feature request, or query
  - May not know which of these until analysis is done, so track in the same database (Bugzilla, github)
- Provides a basis for measurement
- Provides a basis for division of effort
- Facilitates communication
  - Organized record for each issue
  - Ensures problems are not forgotten

----- Comment #4 From [Clare Carty](#) 2006-10-11 15:28 [reply] -----  
(In reply to [comment #3](#))  
> I'm sorry but we really don't have enough details to be able  
> problem. Could you try with another VM?  
>  
Problem didn't happen with another JRE - just the sun JRE.

----- Comment #5 From [Oleg Besedin](#) 2006-10-11 15:38 [reply] -----  
This looks like a duplicate of the [bug 92250](#). Could you try if  
with -XX:MaxPermSize=256m ?

----- Comment #6 From [Pascal Rapicault](#) 2006-10-12 12:57 [reply] -----  
After further investigation, setting the permgenspace to 1024 m  
problem.  
\*\*\* This bug has been marked as a duplicate of [92250](#) \*\*\*

----- Comment #7 From [Clare Carty](#) 2006-10-12 15:18 [reply] -----  
This problem is still occurring on the dependent product with M  
to 1024M. Please investigate.

Bug List: (48 of 200) [First](#) [Last](#) [Prev](#) [Next](#) [Show last search results](#) [Search page](#) [Enter new bug](#)

[Eclipse] Bug#: [160502](#) Hardware:  Reporter: [Clare Carty](#)  
Product:  OS:  [<ccarty@ca.ibm.com>](#)  
Component:  Version:  Add CC:   
Status: REOPENED Priority:  CC: [ccarty@ca.ibm.com](#)  
Resolution:  Severity:  [john\\_arthorne@ca.ibm.com](#)  
Assigned To:  Target:  Milestone:  ☐ Remove selected CCs

QA Contact:   
URL:   
Summary:   
Status:   
Whiteboard:   
Keywords:

Attachment	Type	Created	Size	Actions
<a href="#">screenshot of crash</a>	image/jpeg	2006-10-11 12:14	131.55 KB	<a href="#">Edit</a>
<a href="#">Create a New Attachment</a> (proposed patch, testcase, etc.)				<a href="#">View All</a>

Bug 160502 depends on:  [Show dependency tree](#)  
Bug 160502 blocks:

Votes: 0 [Show votes for this bug](#) [Vote for this bug](#)

# Bug Tracking on GitHub

- Every GitHub project has own issue tracker (and wiki); enable in project settings

The screenshot shows the GitHub interface for the repository 'ckaestne / TypeChef'. The top navigation bar includes the GitHub logo, a search bar, and links to 'Explore', 'Gist', 'Blog', and 'Help'. The repository name 'ckaestne / TypeChef' is displayed, along with buttons for 'Unwatch' (6), 'Unstar' (15), and 'Fork' (11). The 'Issues' tab is selected, showing a list of 16 open issues. The issues are sorted by 'Newest' and are categorized by 'Everyone's Issues' (16), 'Assigned to you' (2), 'Created by you' (14), and 'Mentioning you' (0). The issues list includes:

- Liveness** (#27) - Opened by joliebig 2 days ago, 3 comments.
- Handling of labled statements broken** (#21) - Opened by ckaestne 8 months ago, 1 comment. Label: CParser.
- CParser creates dead nodes** (#20) - Opened by joliebig 9 months ago, 1 comment. Label: CParser.
- Support modules in Linux** (#19) - Opened by ckaestne 10 months ago. Labels: FeatureModelExtraction, CLinker.
- Normalize signatures before linking** (#18) - Opened by ckaestne 10 months ago. Label: CLinker.

The left sidebar shows the 'Labels' section with the following counts:

- CLinker: 3
- CParser: 4
- FeatureModelExtraction: 4

# Formal Verification of Object-Oriented Programs



# Formal Verification

- Proving the correctness of an implementation with respect to a formal specification, using formal methods of mathematics.
- Formally prove that all possible executions of an implementation fulfill the specification
- Manual effort; partial automation; not automatically decidable

# Formal Specifications

```
/*@ requires len >= 0 && array != null && array.Length == len;  
@  
@ ensures \result ==  
@         (\sum int j; 0 <= j && j < len; array[j]);  
@*/  
int total(int array[], int len);
```

Advantage of formal specifications:

- \* runtime checks for free
- \* basis for formal verification
- \* assisting automatic analysis tools

JML (Java Modelling Language)  
as specifications language in  
Java (inside comments)

## Recap: Hoare-Style Verification

- Formal reasoning about program correctness using pre- and postconditions
- Syntax:  $\{P\} S \{Q\}$ 
  - P and Q are predicates
  - P is the precondition
  - S is a program
  - Q is the postcondition
- Semantics
  - If we start in a state where P is true and execute S, then S will terminate in a state where Q is true

# Recap: Hoare-Logic Rules

## Assignments

$$\{ P[E/x] \} x := E \{ P \}$$

## Composition

$$\frac{\{ P \} S \{ Q \} \quad \{ Q \} T \{ R \}}{\{ P \} S; T \{ R \}}$$

## If statement

$$\frac{\{ B \ \& \ P \} S \{ Q \} \quad \{ !B \ \& \ P \} T \{ Q \}}{\{ P \} \text{ if } (B) S \text{ else } T \{ Q \}}$$

## While loop with loop invariant P

$$\frac{\{ P \ \& \ B \} S \{ P \}}{\{ P \} \text{ while } (B) S \{ !B \ \& \ P \}}$$

## Consequence

$$\frac{P \rightarrow P' \quad \{ P \} S \{ Q \} \quad Q \rightarrow Q'}{\{ P' \} S \{ Q' \}}$$

# Hoare Triples – Examples

- $\{ \text{true} \quad \} x := 5 \{ \quad \}$
- $\{ \quad \} x := x + 3 \{ x = y + 3 \quad \}$
- $\{ \quad \} x := x * 2 + 3 \{ x > 1 \quad \}$
- $\{ x=a \quad \} \text{if } (x < 0) \text{ then } x := -x \{ \quad \}$
- $\{ \text{false} \quad \} x := 3 \{ \quad \}$
- $\{ x < 0 \quad \} \text{while } (x \neq 0) \text{ } x := x-1 \{ \quad \}$

# Hoare Triples – Examples

- $\{ \text{true} \quad \} x := 5 \{ x=5 \quad \}$
- $\{ x = y \quad \} x := x + 3 \{ x = y + 3 \quad \}$
- $\{ x > -1 \quad \} x := x * 2 + 3 \{ x > 1 \quad \}$
- $\{ x=a \quad \} \text{if } (x < 0) \text{ then } x := -x \{ x=|a| \quad \}$
- $\{ \text{false} \quad \} x := 3 \{ x = 8 \quad \}$
- $\{ x < 0 \quad \} \text{while } (x \neq 0) x := x-1 \{ \quad \}$ 
  - no such triple!

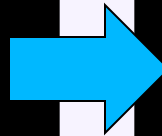
# Recap: 122 midterm

```
int find_peak_bin(int[] A, int n)
//@requires 0 < n && n <= \length(A);
//@requires is_peaked(A, 0, n);
//@ensures 0 <= \result && \result < n;
//@ensures gt_seg(A[\result], A, 0, \result);
//@ensures gt_seg(A[\result], A, \result+1, n);
{
  int lower = 0;
  int upper = n-1;
  while (lower < upper)
    //@loop_invariant _____ ;
    //@loop_invariant _____ ;
  {
    int mid = lower + (upper-lower)/2;
    //@assert _____ ; /* optional */
    if (A[mid] < A[mid+1])
      lower = mid+1;
    else //@assert _____ ; /* optional */
      upper = mid;
  }
  //@assert _____ ; /* optional */
  return lower;
}
```

# Class Invariants

- Properties about the fields of an object
- Established by the constructor
- Should always hold before and after execution of public methods
- May be invalidated temporarily during method execution

```
public class SimpleSet {  
    int contents[];  
    int size;  
  
    //@ ensures sorted(contents);  
    SimpleSet(int capacity) { ... }  
  
    //@ requires sorted(contents);  
    //@ ensures sorted(contents);  
    boolean add(int i) { ... }  
  
    //@ requires sorted(contents);  
    //@ ensures sorted(contents);  
    boolean contains(int i) { ... }  
}
```



```
public class SimpleSet {  
    int contents[];  
    int size;  
  
    //@invariant sorted(contents);  
    SimpleSet(int capacity) { ... }  
    boolean add(int i) { ... }  
    boolean contains(int i) { ... }  
}
```



# Behavioral Subtyping (Liskov Substitution Principle)

Let  $q(x)$  be a property provable about objects  $x$  of type  $T$ . Then  $q(y)$  should be provable for objects  $y$  of type  $S$  where  $S$  is a subtype of  $T$ .

Barbara Liskov

- An object of a subclass should be substitutable for an object of its superclass
- Known already from types:
  - May use subclass instead of superclass
  - Subclass can add, but not remove methods
  - Overriding method must return same or subtype
  - Overriding method may not throw additional exceptions
- Applies more generally to behavior:
  - A subclass must fulfill all contracts that the superclass does
  - Same or stronger invariants
  - Same or **stronger** postconditions for all methods
  - Same or **weaker** preconditions for all methods

# Behavioral Subtyping (Liskov Substitution Principle)

```
abstract class Vehicle {  
    int speed, limit;  
    //@ invariant speed < limit;  
  
    //@ requires speed != 0;  
    //@ ensures |speed| < |\old{speed}|  
    void break();  
}
```

```
class Car extends Vehicle {  
    int fuel;  
    boolean engineOn;  
    //@ invariant fuel >= 0;  
  
    //@ requires fuel > 0 && ! engineOn;  
    //@ ensures engineOn;  
    void start() { ... }  
  
    void accelerate() { ... }  
  
    //@ requires speed != 0;  
    //@ ensures |speed| < |\old{speed}|  
    void break() { ... }  
}
```

**Subclass fulfills the same invariants (and additional ones)**  
**Overridden method has the same pre and postconditions**

# Behavioral Subtyping (Liskov Substitution Principle)

```
class Car extends Vehicle {  
    int fuel;  
    boolean engineOn;  
    //@ invariant fuel >= 0;  
  
    //@ requires fuel > 0 && ! engineOn;  
    //@ ensures engineOn;  
    void start() { ... }  
  
    void accelerate() { ... }  
  
    //@ requires speed != 0;  
    //@ ensures |speed| < |old{speed}|  
    void break() { ... }  
}
```

```
class Hybrid extends Car {  
    int charge;  
    //@ invariant charge >= 0;  
  
    //@ requires (charge > 0 || fuel > 0)  
                && ! engineOn;  
    //@ ensures engineOn;  
    void start() { ... }  
  
    void accelerate() { ... }  
  
    //@ requires speed != 0;  
    //@ ensures |speed| < |old{speed}|  
    //@ ensures charge > old{charge}  
    void break() { ... }  
}
```

**Subclass fulfills the same invariants (and additional ones)**  
**Overridden method start has weaker precondition**  
**Overridden method break has stronger postcondition**

# Behavioral Subtyping (Liskov Substitution Principle)

```
class Rectangle {  
    int h, w;  
  
    Rectangle(int h, int w) {  
        this.h=h; this.w=w;  
    }  
  
    //methods  
}
```

```
class Square extends Rectangle {  
    Square(int w) {  
        super(w, w);  
    }  
}
```

**Is Square a behavior subtype of Rectangle?**

# Behavioral Subtyping (Liskov Substitution Principle)

```
class Rectangle {  
    //@ invariant h>0 && w>0;  
    int h, w;  
  
    Rectangle(int h, int w) {  
        this.h=h; this.w=w;  
    }  
  
    //methods  
}
```

```
class Square extends Rectangle {  
    //@ invariant h==w;  
    Square(int w) {  
        super(w, w);  
    }  
}
```

**Is Square a behavior subtype of Rectangle?**

# Behavioral Subtyping (Liskov Substitution Principle)

```
class Rectangle {  
    //@ invariant h>0 && w>0;  
    int h, w;  
  
    Rectangle(int h, int w) {  
        this.h=h; this.w=w;  
    }  
  
    void scale(int factor) {  
        w=w*factor;  
        h=h*factor;  
    }  
}  
  
class Square extends Rectangle {  
    //@ invariant h==w;  
    Square(int w) {  
        super(w, w);  
    }  
}
```

**Is Square a behavior subtype of Rectangle?**

# Behavioral Subtyping (Liskov Substitution Principle)

```
class Rectangle {  
    //@ invariant h>0 && w>0;  
    int h, w;  
  
    Rectangle(int h, int w) {  
        this.h=h; this.w=w;  
    }  
  
    void scale(int factor) {  
        w=w*factor;  
        h=h*factor;  
    }  
  
    void setWidth(int neww) {  
        w=neww;  
    }  
}
```

```
class Square extends Rectangle {  
    //@ invariant h==w;  
    Square(int w) {  
        super(w, w);  
    }  
}
```

**Is Square a behavior subtype of Rectangle?**

# Behavioral Subtyping (Liskov Substitution Principle)

```
class Rectangle {  
    //@ invariant h>0 && w>0;  
    int h, w;  
  
    Rectangle(int h, int w) {  
        this.h=h; this.w=w;  
    }  
  
    void scale(int factor) {  
        w=w*factor;  
        h=h*factor;  
    }  
  
    void setWidth(int neww)  
        w=neww;  
}
```

```
class Square extends Rectangle {  
    //@ invariant h==w;  
    Square(int w) {  
        super(w, w);  
    }  
}
```

```
class GraphicProgram {  
    void scaleW(Rectangle r, int factor) {  
        r.setWidth(r.getWidth() * factor);  
    }  
}
```

**With these methods, Square is not  
a behavior subtype of Rectangle**



# Formal Verification of Object-Oriented Programs

- Analogue to verification of imperative programs
- Class invariants simplify specifications
- Behavioral subtyping ensures substitutability
- Proof of correctness
  - All possible executions will fulfill the formal specifications
  - Pen and paper proof
  - Support for partially automated proofs available (full automation not possible)

# Static Analysis

# Stupid Bugs

```
public class CartesianPoint {  
    private int x, y;  
    int getX() { return this.x; }  
    int getY() { return this.y; }  
    boolean equals(CartesianPoint that) {  
        return (this.getX()==that.getX()) &&  
            (this.getY() == that.getY());  
    }  
}
```

# FindBugs

The screenshot shows an IDE window with a Java file named `CartesianPoint.java`. The code defines an `equals` method for the `CartesianPoint` class. Below the code, the FindBugs tool has identified two warnings. The first warning, 'FindBugs Problem (Of concern)', states that `CartesianPoint` defines `equals` and uses `Object.hashCode()`. The second warning, 'FindBugs Problem (Scary)', states that `CartesianPoint` defines `equals(CartesianPoint)` method and uses `Object.equals(Object)`. The 'Bug Info' panel provides a detailed explanation of the second warning, noting that the class defines a covariant version of the `equals()` method but inherits the normal `equals(Object)` method from the base `java.lang.Object` class. It suggests that the class should probably define a `boolean equals(Object)` method. The bug is categorized as 'Correctness' with a 'Scary' rank.

```
public boolean equals(CartesianPoint p) {  
    return (p.x==this.x) && (p.y==this.y);  
}
```

0 errors, 2 warnings, 0 others

Description	Resou
FindBugs Problem (Of concern) (1 item)	
CartesianPoint defines equals and uses Object.hashCode()	Cartes
FindBugs Problem (Scary) (1 item)	
CartesianPoint defines equals(CartesianPoint) method and uses Object.equals(Object)	Cartes

**Bug Info**

CartesianPoint.java: 12

Navigation

CartesianPoint defines equals(CartesianPoint) method and uses Object.equals(Object)

**Bug:** CartesianPoint defines equals(CartesianPoint) method and uses Object.equals(Object)

This class defines a covariant version of the `equals()` method, but inherits the normal `equals(Object)` method defined in the base `java.lang.Object` class. The class should probably define a `boolean equals(Object)` method.

**Confidence:** Normal, **Rank:** Scary (8)  
**Pattern:** EQ\_SELF\_USE\_OBJECT  
**Type:** Eq, **Category:** CORRECTNESS (Correctness)

# CheckStyle

The screenshot shows an IDE window titled 'CartesianPoint.java' containing the following Java code:

```
public final class CartesianPoint {  
    private int X,Y;  
    CartesianPoint(int x, int y) {  
        this.X=x;  
        this.Y = y;  
    }  
  
    public int GetY() {  
        return Y;  
    }  
  
    public int getX() {  
        return X;  
    }  
}
```

On the right, the 'Task List' and 'Outline' panels are visible. The 'Task List' panel shows a 'Connect Mylyn' button and a link to 'Connect to your task and ALM tools or create a new task'. The 'Outline' panel shows the 'CartesianPoint' class with fields 'X:int' and 'Y:int'.

At the bottom, the 'Pro' panel displays the status '0 errors, 9 warnings, 0 others'. Below this, a table lists the warnings:

Description	Resolution
▼ Checkstyle Problem (9 items)	
';' is not followed by whitespace.	Carte
'=' is not followed by whitespace.	Carte
'=' is not preceded with whitespace.	Carte
File contains tab characters (this is the first instance).	Carte
Name 'GetY' must match pattern '^[a-z][a-zA-Z0-9]*\$'.	Carte
Name 'X' must match pattern '^[a-z][a-zA-Z0-9]*\$'.	Carte
Name 'Y' must match pattern '^[a-z][a-zA-Z0-9]*\$'.	Carte

# Static Analysis

- Analyzing code without executing it (automated inspection)
- Looks for bug patterns
- Attempts to formally verify specific aspects
- Point out typical bugs or style violations
  - NullPointerExceptions
  - Incorrect API use
  - Forgetting to close a file/connection
  - Concurrency issues
  - And many, many more (over 250 in FindBugs)
- Integrated into IDE or build process
- FindBugs and CheckStyle open source, many commercial products exist

## Example FindBugs Bug Patterns

- Correct equals()
- Use of ==
- Closing streams
- Illegal casts
- Null pointer dereference
- Infinite loops
- Encapsulation problems
- Inconsistent synchronization
- Inefficient String use
- Dead store to variable

# Bug finding

```
public Boolean decide() {  
    if (computeSomething()==3)  
        return Boolean.TRUE;  
    if (computeSomething()==4)  
        return false;  
    return null;  
}
```

Problem @ Javadoc Declarati Search Console Coverag History Bug Info Bug Expl

A.java: 69

Navigation

**Bug:** FBTest.decide() has Boolean return type and returns explicit null

A method that returns either Boolean.TRUE, Boolean.FALSE or null is an accident waiting to happen. This method can be invoked as though it returned a value of type boolean, and the compiler will insert automatic unboxing of the Boolean value. If a null value is returned, this will result in a NullPointerException.

**Confidence:** Normal, **Rank:** Troubling (14)

**Pattern:** NP\_BOOLEAN\_RETURN\_NULL

**Type:** NP, **Category:** BAD\_PRACTICE (Bad practice)



# Abstract Interpretation

- Static program analysis is the **systematic examination** of an **abstraction of a program's state space**
- Abstraction
  - Don't track everything! (That's normal interpretation)
  - Track an important abstraction
- Systematic
  - Ensure everything is checked in the same way

**Details on how this works in 15-313**

# Comparing Quality Assurance Strategies

	Error exists	No error exists
Error Reported	True positive (correct analysis result)	False positive (annoying noise)
No Error Reported	False negative (false confidence)	True negative (correct analysis result)

Sound Analysis:

reports all defects

-> no false negatives

typically overapproximated

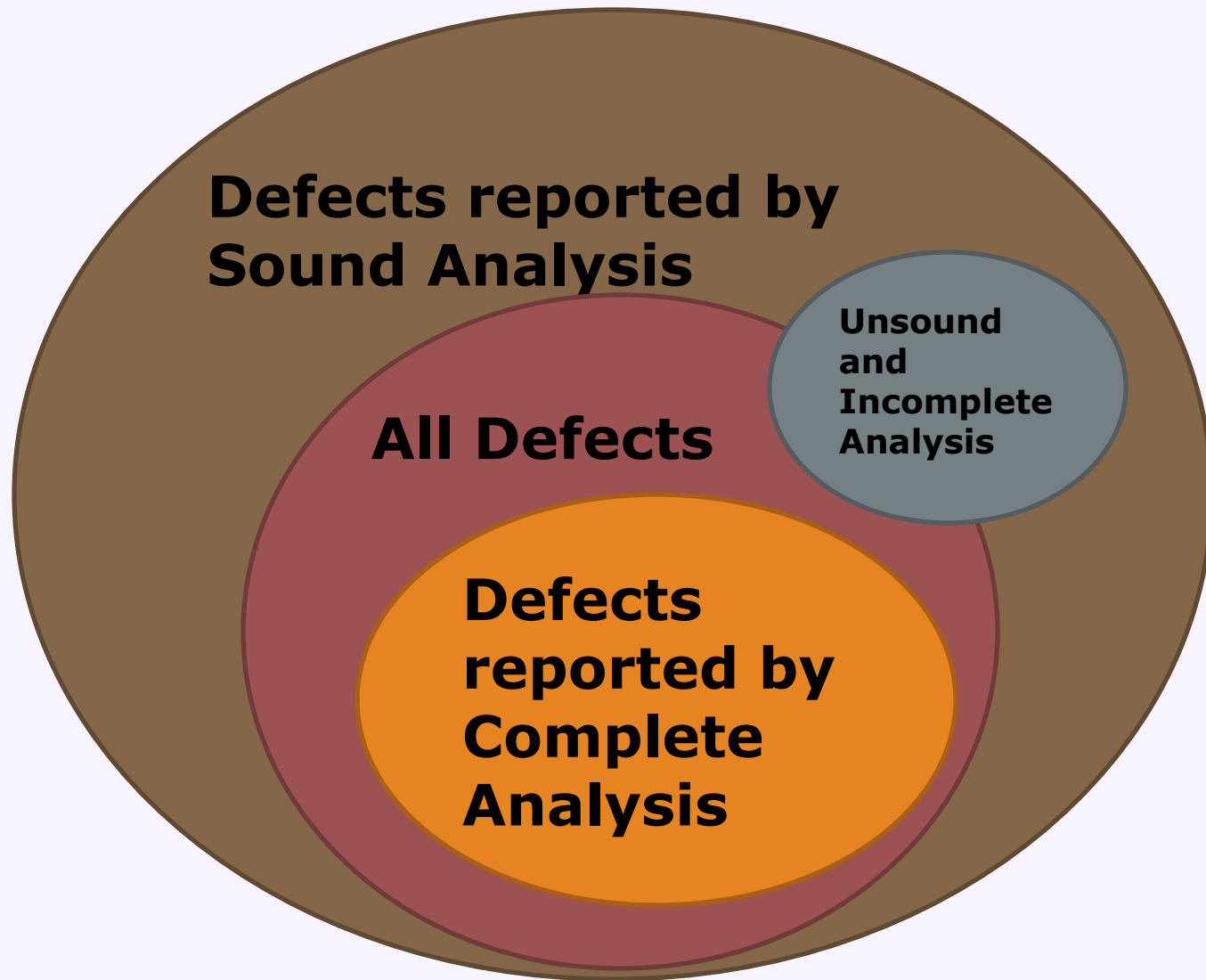
Complete Analysis:

every reported defect is an actual defect

-> no false positives

typically underapproximated

## How does testing relate? And formal verification?



**"Any nontrivial property about the language recognized by a Turing machine is undecidable."**

**Henry Gordon Rice, 1953**

- Every static analysis is necessarily incomplete or unsound or undecidable (or multiple of these)
- Each approach has different tradeoffs

# Soundness / Completeness / Performance Tradeoffs

- Type checking does catch a specific class of problems (sound), but does not find all problems
- Compiler optimizations must err on the safe side (only perform optimizations when sure it's correct; -> complete)
- Many practical bug-finding tools analyses are unsound and incomplete
  - Catch typical problems
  - May report warnings even for correct code
  - May not detect all problems
- Overwhelming amounts of false negatives make analysis useless
- Not all "bugs" need to be fixed

# Testing and Proofs

- Testing

- Observable properties
- Verify program for one execution
- Manual development with automated regression
- Most practical approach now
- Does not find all problems (unsound)

- Proofs (Formal Verification)

- Any program property
- Verify program for all executions
- Manual development with automated proof checkers
- Practical for small programs, may scale up in the future
- Sound and complete, but not automatically decidable

- So why study proofs if they aren't (yet) practical?
  - Proofs tell us how to think about program correctness
  - Important for development, inspection, dynamic assertions
  - Foundation for static analysis tools
  - These are just simple, automated theorem provers
  - Many are practical today!

# Testing, Static Analysis, and Proofs

- Testing

- Observable properties
- Verify program for one execution
- Manual development with automated regression
- Most practical approach now
- Does not find all problems (unsound)

- Static Analysis

- Analysis of all possible executions
- Specific issues only with conservative approx. and bug patterns
- Tools available, useful for bug finding
- Automated, but unsound and/or incomplete

- Proofs (Formal Verification)

- Any program property
- Verify program for all executions
- Manual development with automated proof checkers
- Practical for small programs, may scale up in the future
- Sound and complete, but not automatically decidable

**What strategy to use in your project?**



## Quality Assurance Summary

- Reporting and tracking bugs/issues
- Select a quality assurance strategy for functional correctness
- Testing can find faults in specific executions
- Formal verification (Hoare-style pre/post-conditions) can ensure correctness of all executions
  - Class Invariants and Behavioral Subtyping
- Static analysis can find issues for classes of problems
- Soundness vs. Completeness vs. Automation