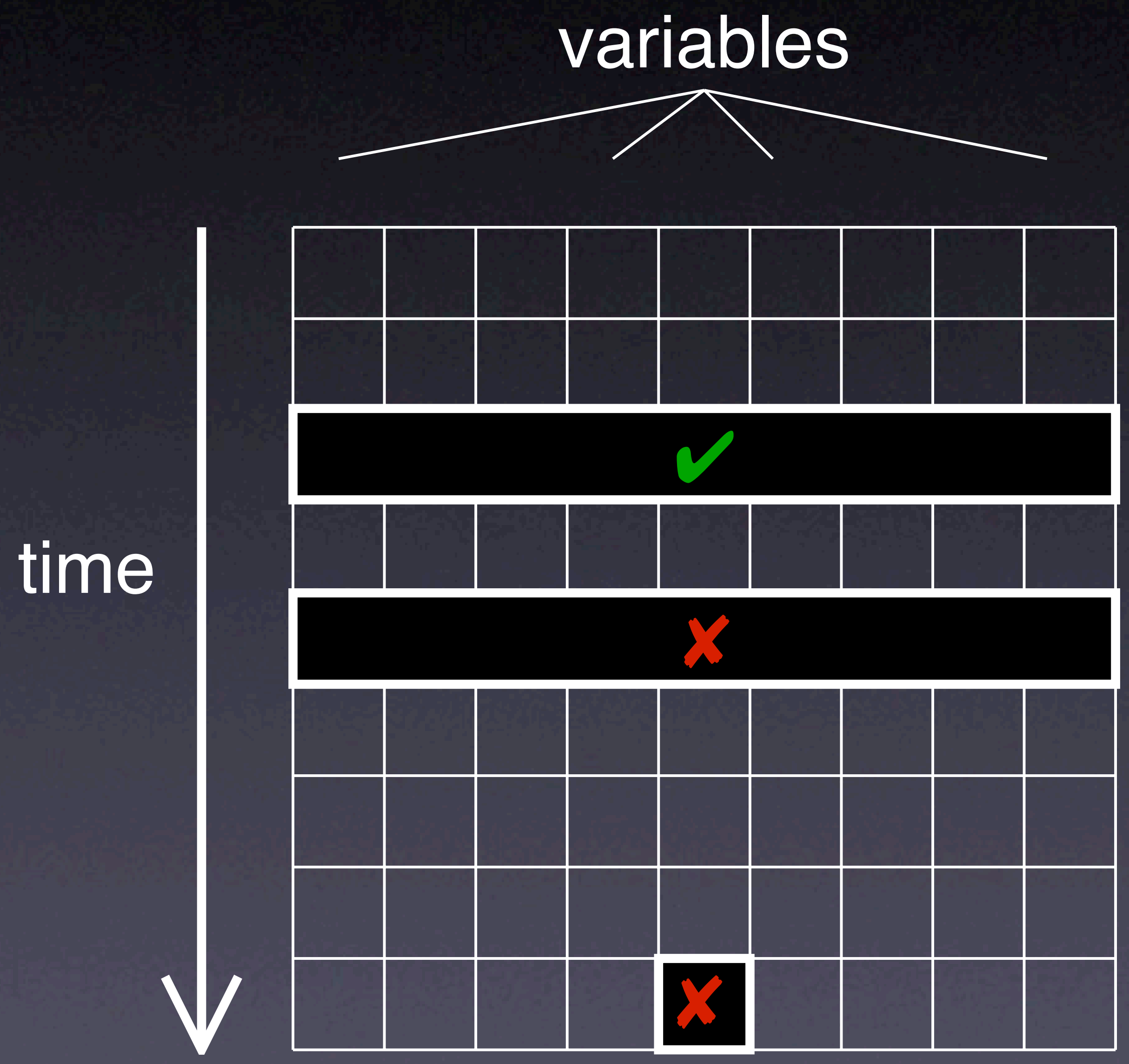
Asserting Expectations

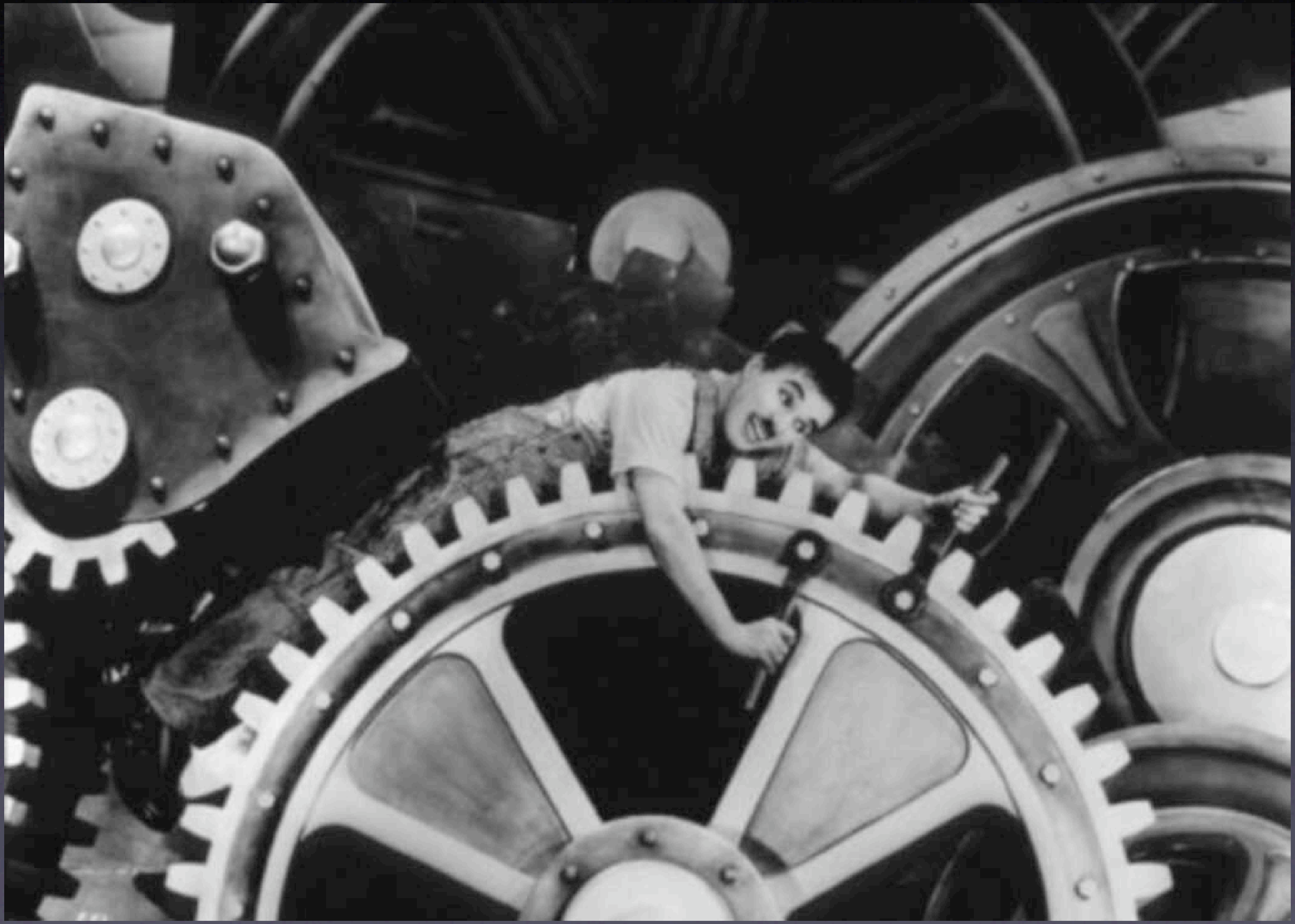
Andreas Zeller

Search in Time

- During execution, the state becomes infected.
- Basic idea: Observe a *transition* from sane to infected.



Manual Observation



Automated Observation

Automated Observation

what to observe	when to observe	what to expect

Basic Assertions

```
if (divisor == 0) {
    printf("Division by zero!");
    abort();
}
```

Specific Assertions

```
assert (divisor != 0);
```

Implementation

```
void assert (int x)
{
  if (!x)
  {
    printf("Assertion failed!\n");
    abort();
  }
}
```

Execution

```
$ my-program
Assertion failed!
Abort (core dumped)
$
```

Better Diagnostics

```
$ my-program
divide.c:37:
   assertion ‘divisor != 0’ failed
Abort (core dumped)
$ _
```

Assertions as Macros

```
#ifndef NDEBUG
#define assert(ex) \
((ex) ? 1 : (cerr << __FILE__ << ":" << __LINE__ \
            << ": assertion ‘" #ex "’ failed\n", \
                  abort(), 0))
#else
#define assert(x) ((void) 0)
#endif
```

Automated Observation

what to observe	**when** to observe	**what** to expect
state checked in assertion	location of assertion	checked property of program state

When to observe

- Data invariants
- Pre- and postconditions

Asserting Invariants

A Time Class

```
class Time {
public:
    int hour();      // 0..23
    int minutes();   // 0..59
    int seconds();   // 0..60 (incl. leap seconds)

    void set_hour(int h);
    …
}
```

Any time from 00:00:00 to 23:59:60 is valid

Ensuring Sanity

```
void Time::set_hour(int h)
{
    // precondition
    assert (0 <= hour() && hour() <= 23) &&
           (0 <= minutes() && minutes() <= 59) &&
           (0 <= seconds() && seconds() <= 60);
    …
    // postcondition
    assert (0 <= hour() && hour() <= 23) &&
           (0 <= minutes() && minutes() <= 59) &&
           (0 <= seconds() && seconds() <= 60);
}
```

Ensuring Sanity

```
bool Time::sane()
{
    return (0 <= hour() && hour() <= 23) &&
           (0 <= minutes() && minutes() <= 59) &&
           (0 <= seconds() && seconds() <= 60);
}

void Time::set_hour(int h)
{
    assert (sane());  // precondition
    …
    assert (sane());  // postcondition
}
```

Ensuring Sanity

```
bool Time::sane()
{
    return (0 <= hour() && hour() <= 23) &&
           (0 <= minutes() && minutes() <= 59) &&
           (0 <= seconds() && seconds() <= 60);
}
```

sane() is the *invariant* of a Time object:

- holds *before* every public method
- holds *after* every public method

Ensuring Sanity

```
bool Time::sane()
{
    return (0 <= hour() && hour() <= 23) &&
           (0 <= minutes() && minutes() <= 59) &&
           (0 <= seconds() && seconds() <= 60);
}

void Time::set_hour(int h)
{
    assert (sane());
    …
    assert (sane());
}
```

same for set_minute(), set_seconds(), etc.

Locating Infections

- Precondition failure = infection *before* method
- Postcondition failure = infection *within* method
- All assertions pass = no infection

```
void Time::set_hour(int h)
{
    assert (sane());  // precondition
    …
    assert (sane());  // postcondition
}
```

Complex Invariants

```
class RedBlackTree {
  …
  boolean sane() {
      assert (rootHasNoParent());
      assert (rootIsBlack());
      assert (redNodesHaveOnlyBlackChildren());
      assert (equalNumberOfBlackNodesOnSubtrees());
      assert (treeIsAcyclic());
      assert (parentsAreConsistent());

      return true;
  }
}
```

Invariants as Aspects

```
public aspect RedBlackTreeSanity {
    pointcut modify():
        call(void RedBlackTree.add*(..)) ||
        call(void RedBlackTree.del*(..));

    before(): modify() {
        assert (sane());
    }

    after(): modify() {
        assert (sane());
    }
}
```

Invariants in GDB

```
(gdb) break 'Time::set_hour(int)' if !sane()
Breakpoint 3 at 0x2dcf: file Time.C, line 45.
(gdb) _
```

Asserting Correctness

Postconditions

```
def divide(dividend, divisor):
    # Actual computation goes here
    …
    assert quotient * divisor + remainder == dividend
    return (quotient, remainder)
```

Postconditions

```
void Time::set_hour(int h)
{
   // Actual code goes here

   assert (hour() == h);   // postcondition
}
```

Postconditions

```
void Sequence::sort()
{
    // Actual code goes here

    assert (is_sorted());
}
```

helper function

Postconditions

```
void Container::insert(Item x)
{
    // Actual code goes here

    assert (has(x));
}
```

a helper function that is also a useful public method

Postconditions

```
void Heap::merge(Heap another_heap)
{
    assert (sane());
    assert (another_heap.sane());

    // Actual code goes here

    assert (sane());
}
```

Invariants are always part of pre- and postconditions

Checking Earlier State

```
void Time::set_hour(int h)
{
   int old_minutes = minutes();
   int old_seconds = seconds();
   assert (sane());

   // Actual code goes here

   assert (sane());
   assert (hour() == h);
   assert (minutes() == old_minutes &&
           seconds() == old_seconds);
}
```

Contracts

```
set_hour (h: INTEGER) is
        -- Set the hour from `h'
    require
        sane_h: 0 <= h and h <= 23
    ensure
        hour_set: hour = h
        minute_unchanged: minutes = old minutes
        second_unchanged: seconds = old seconds
```

This *contract* specifies interface properties

Z Invariant

Date

$$hours, minutes, seconds : \mathbb{N}$$

$$0 \leq hours \leq 23$$
$$0 \leq minutes \leq 59$$
$$0 \leq seconds \leq 59$$

Z Conditions

$$
\begin{array}{l}
\textit{set\_hour} \\
\hline
\Delta Date \\
h? : \mathbb{N} \\
\hline
0 \leq h? \leq 23 \\
hours' = h? \\
minutes' = minutes \\
seconds' = seconds
\end{array}
$$

Spec vs Code

Contracts

```
set_hour (h: INTEGER) is
        -- Set the hour from `h'
    require
        sane_h: 0 <= h and h <= 23
    ensure
        hour_set: hour = h
        minute_unchanged: minutes = old minutes
        second_unchanged: seconds = old seconds
```

This *contract* specifies interface properties

31

Z Conditions

set_hour

$\Delta Date$

$h? : \mathbb{N}$

$$0 \leq h? \leq 23$$
$$hours' = h?$$
$$minutes' = minutes$$
$$seconds' = seconds$$

33

Integrated spec
limited to language

Separate spec
can express anything

JML

```
/*@ requires 0 <= h && h <= 23
  @ ensures  hours() == h &&
  @          minutes() == \old(minutes()) &&
  @          seconds() == \old(seconds())
  @*/
void Time::set_hour(int h) …
```

Translated into run-time assertions

JML as Spec

```
/*@ requires x >= 0.0;
  @ ensures JMLDouble
  @         .approximatelyEqualTo
  @         (x, \result * \result, eps);
  @*/
```

What does this specify?

```
public class Purse {
  final int MAX_BALANCE;
  int balance;
  //@ invariant  0 <= balance && balance <= MAX_BALANCE;

  byte[] pin;
  /*@ invariant pin != null && pin.length == 4 &&
    @            (\forall int i; 0 <= i && i < 4;
    @                         0 <= byte[i] && byte[i] <= 9)
    @*/

  /*@ requires   amount >= 0;
    @ assignable balance;
    @ ensures    balance == \old(balance) - amount &&
    @            \result == balance;
    @ signals    (PurseException) balance == \old(balance);
    @*/
  int debit(int amount) throws PurseException { … }
```

More use of JML

- Documentation
- Unit testing with JMLUnit
- Invariant generation with DAIKON
- Static checking with ESC/Java
- Verification with theorem provers

Relative Debugging

Rather than checking a spec, we can also compare against a *reference run:*

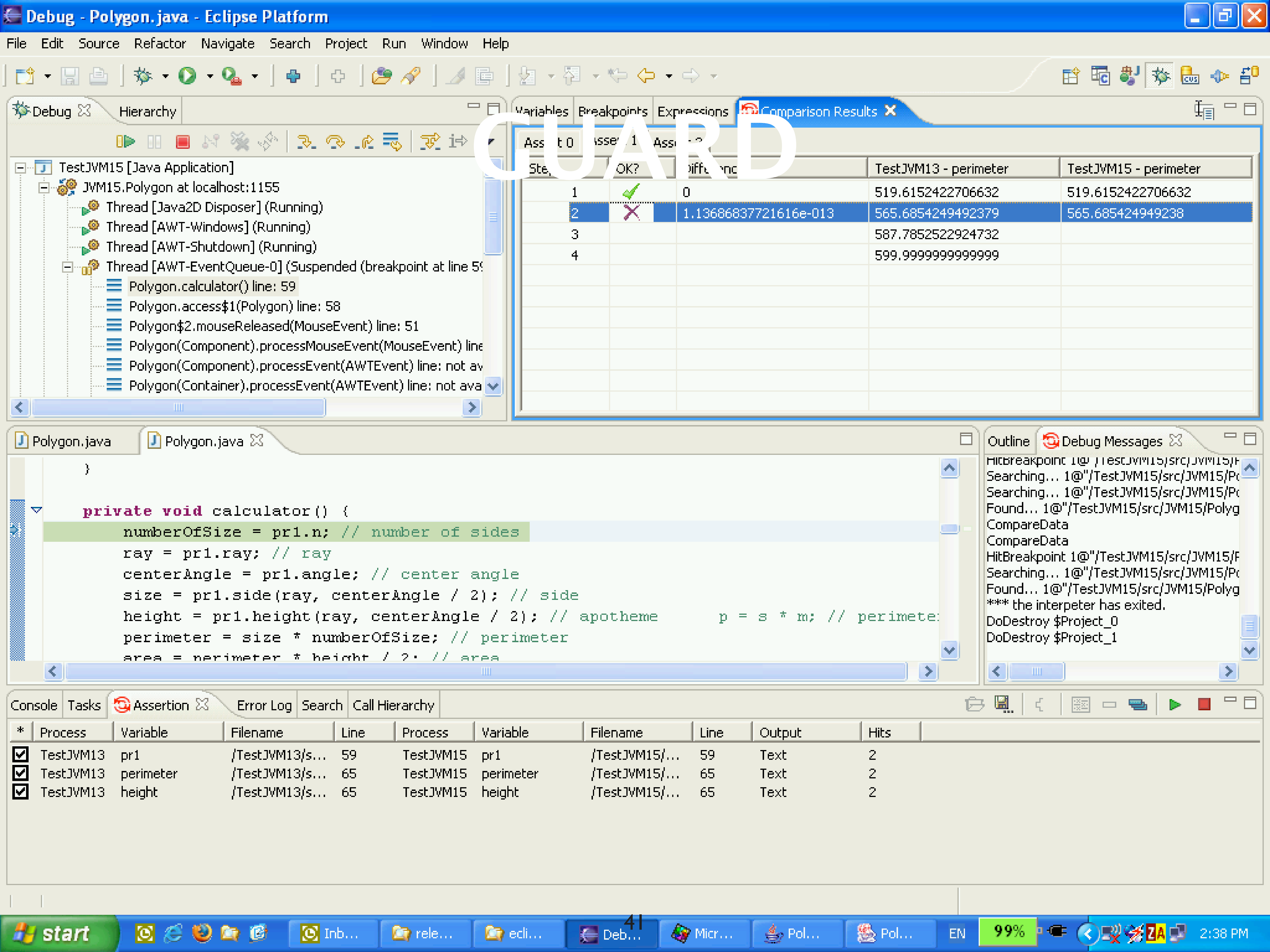
- The environment has changed–e.g. ports or new interpreters
- The code has changed
- The program has been reimplemented

Relative Assertions

- We compare two program runs
- A *relative assertion* compares variable values across the two runs:

```
assert \
p1::perimeter@polygon.java:65 == \
p0::perimeter@polygon.java:65
```

- Specifies when and what to compare


Debug - Polygon.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug Hierarchy
TestJVM15 [Java Application]
JVM15.Polygon at localhost:1155
Thread [Java2D Disposer] (Running)
Thread [AWT-Windows] (Running)
Thread [AWT-Shutdown] (Running)
Thread [AWT-EventQueue-0] (Suspended (breakpoint at line 59
Polygon.calculator() line: 59
Polygon.access$1(Polygon) line: 58
Polygon$2.mouseReleased(MouseEvent) line: 51
Polygon(Component).processMouseEvent(MouseEvent) line
Polygon(Component).processEvent(AWTEvent) line: not av
Polygon(Container).processEvent(AWTEvent) line: not ava
Variables Breakpoints Expressions Comparison Results
GUARD
TestJVM13 - perimeter
TestJVM15 - perimeter
1 0 519.6152422706632 519.6152422706632
2 1.13686837721616e-013 565.6854249492379 565.685424949238
3 587.7852522924732
4 599.9999999999999
Polygon.java Polygon.java
private void calculator() {
numberOfSize = pr1.n; // number of sides
ray = pr1.ray; // ray
centerAngle = pr1.angle; // center angle
size = pr1.side(ray, centerAngle / 2); // side
height = pr1.height(ray, centerAngle / 2); // apotheme p = s * m; // perimete
perimeter = size * numberOfSize; // perimeter
Outline Debug Messages
Searching... 1@"/TestJVM15/src/JVM15/Pc
Found... 1@"/TestJVM15/src/JVM15/Polyg
CompareData
HitBreakpoint 1@"/TestJVM15/src/JVM15/P
*** the interpeter has exited.
DoDestroy $Project_0
DoDestroy $Project_1
Console Tasks Assertion Error Log Search Call Hierarchy
* Process Variable Filename Line Process Variable Filename Line Output Hits
TestJVM13 pr1 /TestJVM13/s... 59 TestJVM15 pr1 /TestJVM15/... 59 Text 2
TestJVM13 perimeter /TestJVM13/s... 65 TestJVM15 perimeter /TestJVM15/... 65 Text 2
TestJVM13 height /TestJVM13/s... 65 TestJVM15 height /TestJVM15/... 65 Text 2
start
2:38 PM

Concepts

- Assertions catch infections before they propagate too far
- Assertions check preconditions, postconditions and invariants
- Assertions can serve as specifications
- A program can serve as reference to be compared against