Validation

David Evans
www.cs.virginia.edu/cs205

Dictionary Definition

val·i·date

1. To declare or make legally valid.
2. To mark with an indication of official sanction.
3. To establish the soundness of; corroborate.

Can we do any of these with software?

Java's License

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

Java's License

5. LIMITATION OF LIABILITY. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. …

Java's License

2. RESTRICTIONS. … Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses.

Software Validation

- Process designed to *increase our confidence* that a program *works as intended*
- For complex programs, cannot often make guarantees
- This is why typical software licenses don't make any claims about their program working

Increasing Confidence

- Testing
  - Run the program on set of inputs and check the results
- Verification
  - Argue formally or informally that the program always works as intended
- Analysis
  - Poor programmer's verification: examine the source code to increase confidence that it works as intended

Testing and Fishing

Using some successful tests to conclude that a program has no bugs, is like concluding there are no fish in the lake because you didn't catch one!

Exhaustive Testing

- Test all possible inputs
- PS1: 50x50 grid, all cells can be either dead or alive before starting

2^{2500} =

3758280234548012036833624189723865048677365517592586770565238397822316814983377085357327257526588443337024577495260577603092278913516177656519073109687802364646940433162365621467244164785911318325937291112215801805317492327775155799689899075142213969117994877343802049421624954402214529390781647563339535024772584901607666862982567918622849636160208877365834950163790188523026247440507390382032188892386109905869706753143243921198482212075444022433366554786856559389689585638126582377224037721702239991441466026185752651502936472280911018500320375496336749951569521541850441747925844066295279671872605285792552660130702047998218334749356321677469529682551765858267502715894007887727250070780350262952377214028842297486263597879792176338220932619489509376

But that's not all: all possible start stop step clicks, different platforms, how long to you need to run it, etc.

Selective Testing

- We can't test everything, pick test cases with high probability of finding flaws
- **Black-Box Testing**: design tests looking only at specification
- **Glass-Box Testing**: design tests looking at code
  - Path-complete: at least one test to exercise each path through code

Black-Box Testing

```
public CellState getNextState ()
    // MODIFIES: this
    // EFFECTS: Returns the next state for this cell.  If a cell is currently
    //    dead cell and has three live neighbors, then it becomes a live cell.
    //    If a cell is currently alive and has two or three live neighbors it
    //    remains alive.   Otherwise, the cell dies.
```

Test all paths through the *specification*


```
public CellState getNextState ()
    // MODIFIES: this
    // EFFECTS: Returns the next state for this cell.  If a cell is currently
    //    dead cell and has three live neighbors, then it becomes a live cell.
    //    If a cell is currently alive and has two or three live neighbors it
    //    remains alive.   Otherwise, the cell dies.
```

Test all paths through the specification:

1. currently dead, three live neighbors
2. currently alive, two live neighbors
3. currently alive, three live neighbors
4. currently dead, < 3 live neighbors
5. currently dead, > 3 live neighbors
6. currently alive, < 2 live neighbors
7. currently alive, > 3 live neighbors

Black-Box Testing

```
public CellState getNextState ()
    // MODIFIES: this
    // EFFECTS: Returns the next state for this cell.  If a cell is currently
    //    dead cell and has three live neighbors, then it becomes a live cell.
    //    If a cell is currently alive and has two or three live neighbors it
    //    remains alive.   Otherwise, the cell dies.
```

Test all (7) paths through the specification

Test boundary conditions

1. all neighbors are dead
2. all neighbors are alive
3. cell is at a corner of the grid
4. cell is at an edge of the grid

Glass-Box Testing

```
public CellState getNextState()
{
  int countalive = 0;
  Enumeration<SimObject> neighbors = getNeighbors();
  while (neighbors.hasMoreElements()) {
    SimObject neighbor = neighbors.nextElement();
    if (neighbor instanceof Cell) {
      Cell cell = (Cell) neighbor;
      if (cell.isAlive()) { countalive++; }
    }
  if (countalive == 3) {
    return CellState.createAlive ();
  } else if (getState ().isAlive () && countalive == 2) {
    return CellState.createAlive ();
  } else { return CellState.createDead (); }
}
```

How many paths are there through this code?

Path-Complete Testing

- Insufficient
  - Often, bugs are missing paths
- Impossible
  - Most programs have an infinite number of paths
  - Loops and recursion
    - Test with zero, one and several iterations
  - Branching
    - Can test all paths

How many paths?

```
if (countalive == 3) {
    return CellState.createAlive ();
  } else if (getState ().isAlive () && countalive == 2) {
    return CellState.createAlive ();
  } else {
    return CellState.createDead ();
  }
}
```


Testing Recap

- Testing can find problems, not to prove your program works
  - Since exhaustive testing is impossible, select test cases with maximum probability of finding bugs
  - A *successful* test case is one that reveals a bug in your program!
- Typically at least 40% of cost of software project is testing, often ~80% of cost for safety-critical software

Quizzing

Testing Recap

- Testing can find problems, but can't prove your program works
  - Since exhaustive testing is impossible, select test cases with maximum probability of finding bugs
  - A *successful* test case is one that reveals a bug in your program!
- If we can't test all possible paths through a program, how can we increase our confidence that it works?

Analysis

- Make claims about *all* possible paths by examining the program code directly, not executing it
- Use formal semantics of programming language to know what things mean
- Use formal specifications of procedures to know that they do

Hopelessness of Analysis

It is impossible to correctly determine if any interesting property is true for an arbitrary program!

The Halting Problem: it is impossible to write a program that determines if an arbitrary program halts.

Compromises

- Use imperfect automated tools:
  - Accept unsoundness and incompleteness
  - **False positives**: sometimes an analysis tool will report warnings for a program, when the program is actually okay (unsoundness)
  - **False negatives**: sometimes an analysis tool will report no warnings for a program, even when the program violates properties it checks (incompleteness)
- **Use informal reasoning**
- **Design programs to modularize reasoning**

Charge

- Next class:
  - ps2 hints
  - Exceptions, programming defensively