Automated Software Engineering
Writing Code to Help You Write Code

Gregory Gay

CSCE 190 - Computing in the Modern World

October 27, 2015

Software Engineering

The development and evolution of **high-quality** (large) software systems in a **systematic, controlled, and efficient** manner.

Necessary because **society depends on software**.

Flawed Software Will Be Exploited

40 Million Card Accounts Affected by Security Breach at Target

Sony: Hack so bad, our computers still don't work

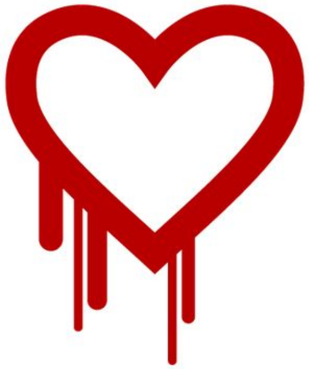
By Charles Riley @CRrileyCNN January 23, 2015: 10:10 AM ET

Recommend 182

The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.



Software Can Hurt People

In 2010, software problems were responsible for **26% of medical device recalls**.

"There is a reasonable probability that use of these products will cause serious adverse health consequences or death."

- **US Food and Drug Administration**

Why Software Engineering?

Good engineering is **difficult and expensive**.

"It costs 50% more per instruction to develop high-dependability software than to develop low-dependability software."

- **Victor Basili (Emeritus Professor, UMD)**

Software engineering is focused on lowering the cost and difficulty, and improving the quality, of software development.

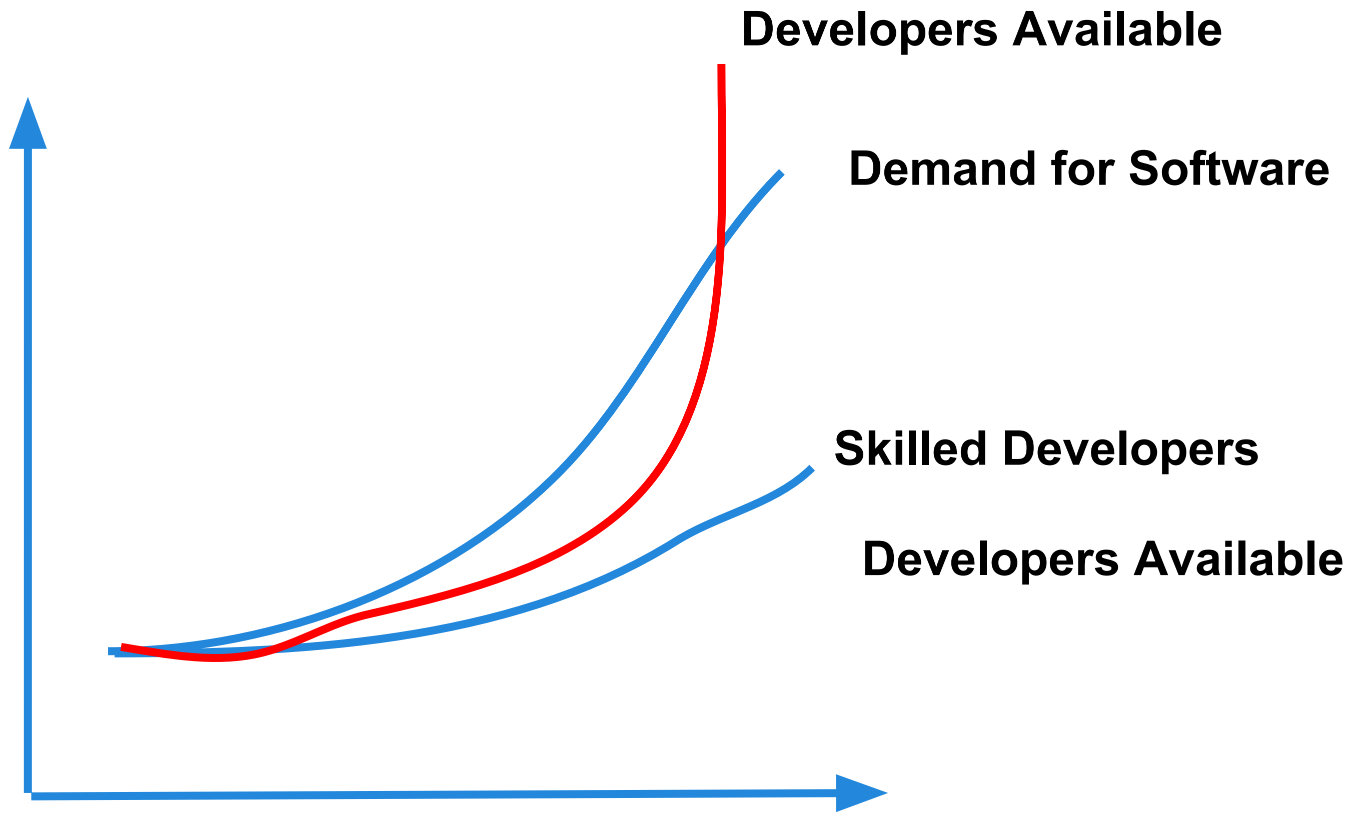
The Need for Disciplined Practices

The job of software engineers is to:

- produce high-quality products
- produce them on schedule
- and do this within planned costs

You should start learning now (you’ll want the practice).

Developers in Demand

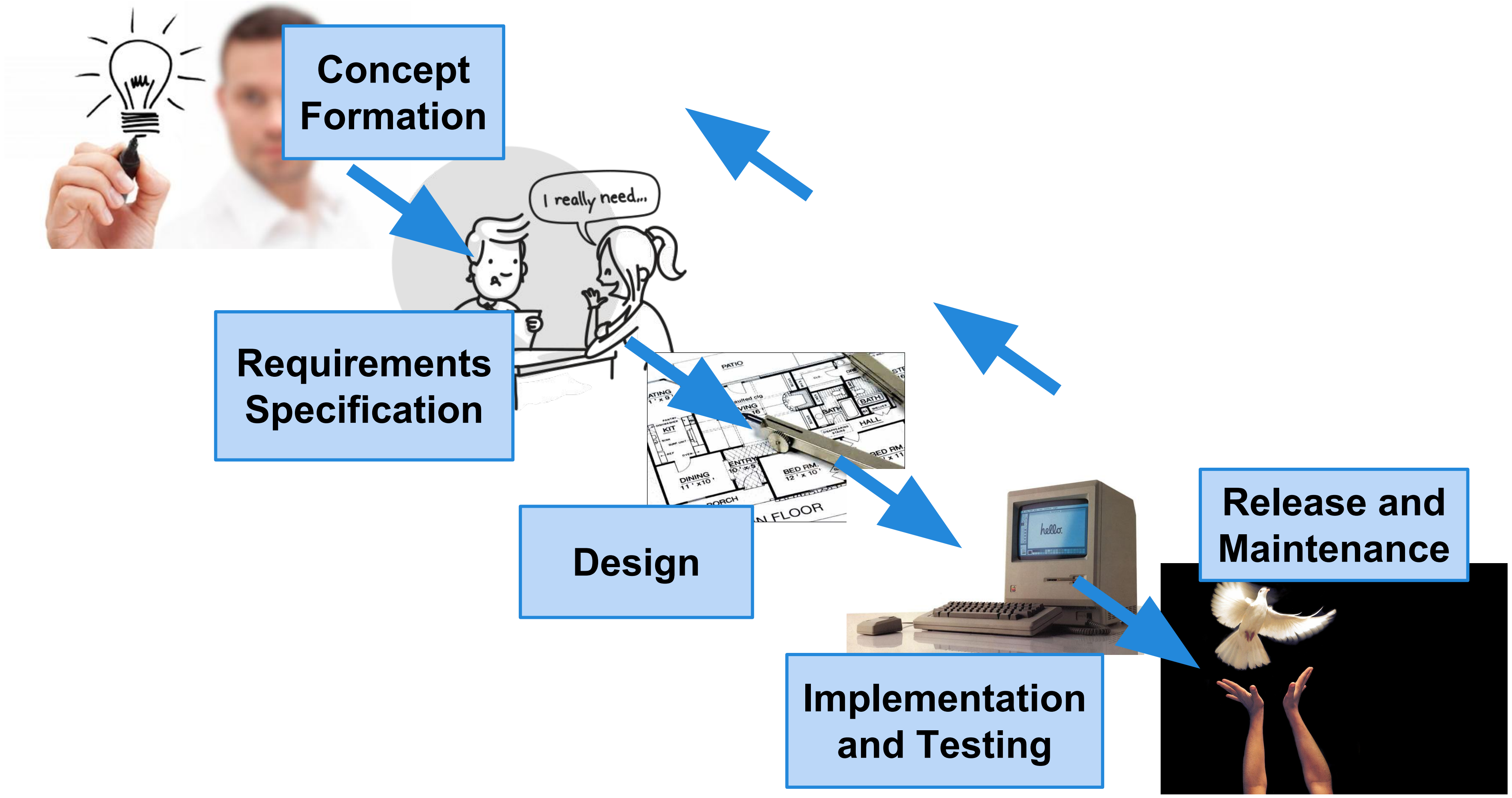


What is the largest programming project you’ve ever worked on?

How did you design and build it?

What are the largest pieces of software in the world?

Typical Development Process



Focus Areas in SE

- Development Processes
  - How people work together to create software.
- Requirements Engineering
  - How to formally describe the properties and expected behavior of the software we will build.
- Software Design and Architecture
  - How to design robust and efficient systems.
- Software Testing & Verification
  - How to ensure that software is correct and free of faults.

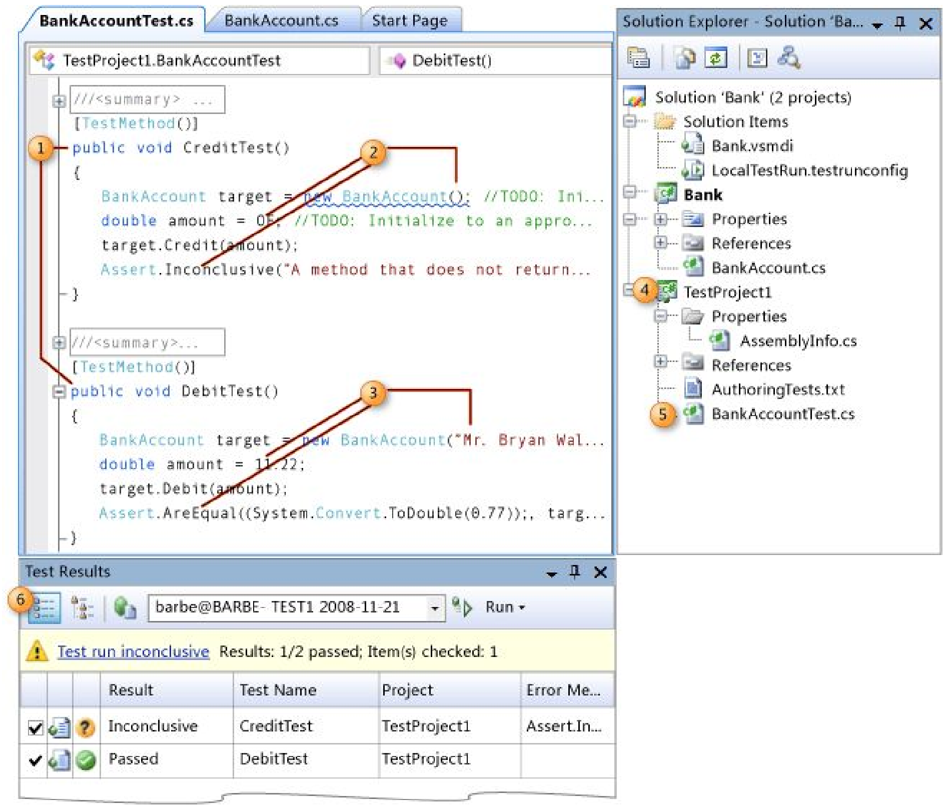
“Why do we need all of this documentation?”
- Nearly Every Student Ever

Automated Software Engineering:

Writing code to help you write code.

Testing Requires Writing Code

- Testing requires writing code to break your code.
- Sometimes, you write more code to test than is in the original software.



```
///<summary> ...
[TestMethod()]
public void CreditTest()
{
    BankAccount target = new BankAccount(); //TODO: Ini...
    double amount = 0F; //TODO: Initialize to an appro...
    target.Credit(amount);
    Assert.Inconclusive("A method that does not return...
}

///<summary>...
[TestMethod()]
public void DebitTest()
{
    BankAccount target = new BankAccount("Mr. Bryan Wal...
    double amount = 11.22;
    target.Debit(amount);
    Assert.AreEqual((System.Convert.ToDouble(0.77));, targ...
}
```

Test run inconclusive Results: 1/2 passed; Item(s) checked: 1

			Result	Test Name	Project	Error Me...
✓		?	Inconclusive	CreditTest	TestProject1	Assert.In...
✓		✓	Passed	DebitTest	TestProject1	

Testing Requires Writing Code

- Unit Tests
  - Instantiates classes.
  - Passes in input to functions.
  - Checks output against expectations.
- Environment Simulation
  - Models physical environment.
  - Experiment with different network and hardware configurations.
- Component Mocking
  - “Fake” behavior for undeveloped classes.

Testing is Expensive

- Testing often claims over 50% of the development time and budget.
- Many parts of testing are labor-intensive:
  - Coming up with input and expected outputs.
  - Tracing a fault to its source in the code.
  - Fixing the code without breaking other code.
  - … (and basically everything else)

Automated Testing: Writing code that writes code that breaks code.

Automated Test Generation

If we can score “test quality”, then we have an **optimization goal**.

Treat test generation as a search problem.

- Generate a test (or set of tests).
- Score each of them by their adequacy.
- Manipulate the population according to a search strategy (a **“heuristic”**).

Metaheuristic Search

Most search spaces are too large to exhaustively explore. Instead, choose a smart strategy to stochastically sample from that space.

- We can't guarantee an optimal solution…
  - … but if we're smart, we'll hit something close enough.
  - Metaheuristic search is computationally feasible on problems where complete search is not.

Local Search

- Generate a potential solution.
- Score it using your fitness function.
- Attempt to improve it by looking at its **local neighborhood**.
  - Keep making small, incremental improvements.
- Very fast and efficient if you make a good initial guess.
- Can get stuck in local maxima if not.
  - Reset strategies help.

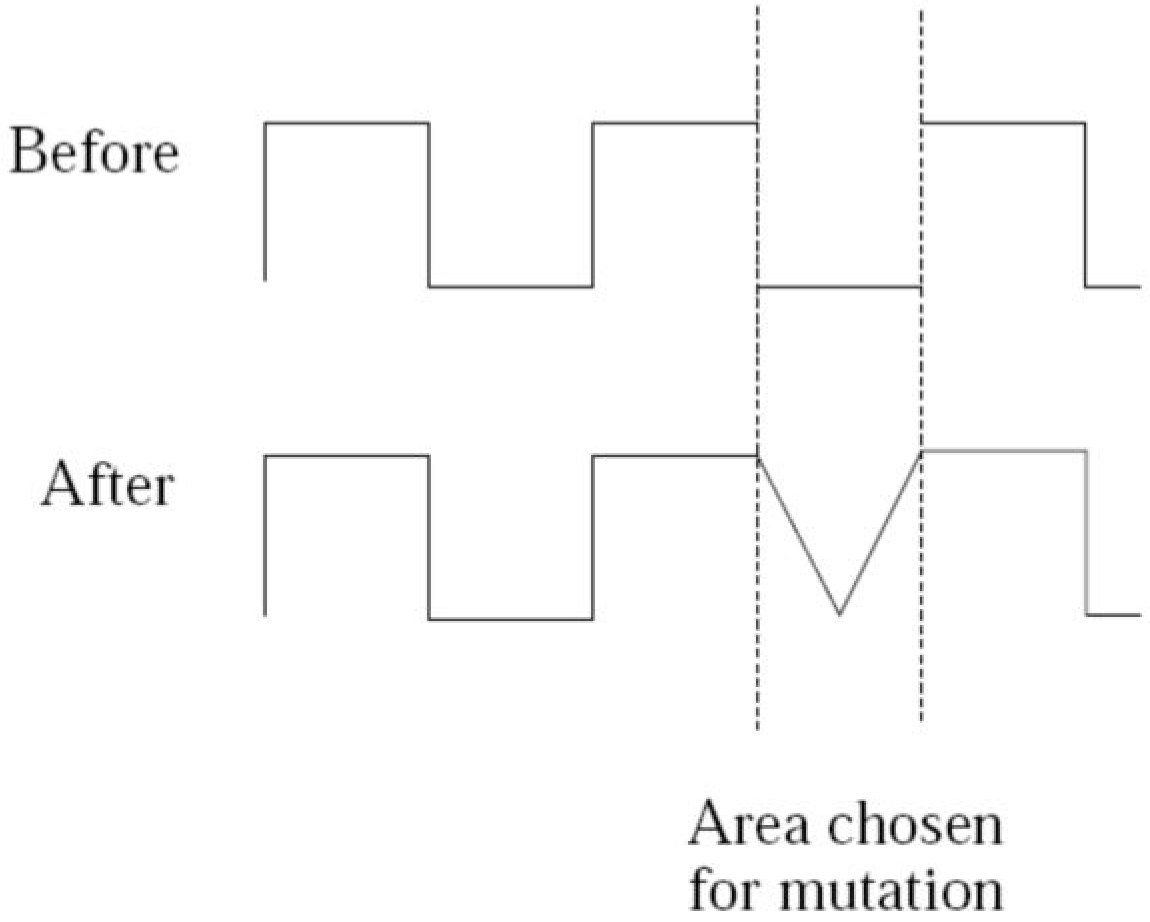
Global Search

- Generate a potential solution (or set of solutions).
- Score them.
- At a certain probability, sample from other regions of the space.
- Strategies typically based on natural processes - swarm attack patterns, ant colony behavior, species evolution.

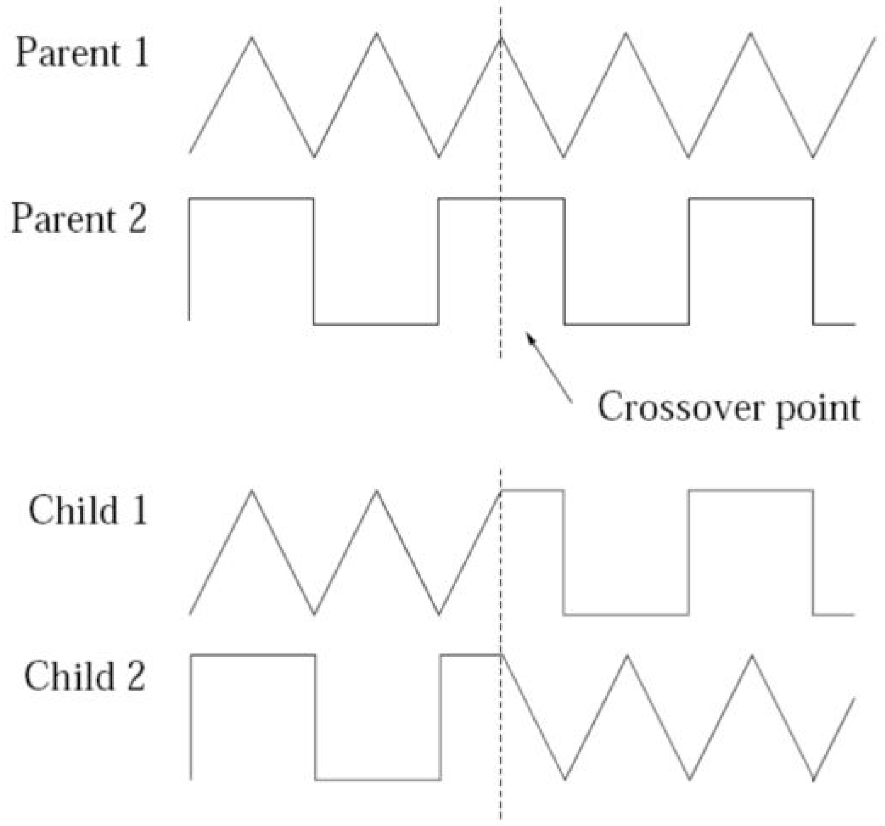
Genetic Algorithms

- Over multiple generations, evolve a population - favoring good solutions and filtering out bad solutions.
- Diversity is introduced to the population each generation by:
  - Keeping some of the best solutions.
  - Randomly generating some population members.
  - Creating “offspring” through mutation and gene crossover.

Genetic Algorithms - Mutation



Genetic Algorithms - Crossover



Fitness Functions

- Solutions are judged by a “fitness function” that takes in the solution and calculates a score.
  - Distance from the current solution to the “ideal” solution.
    - How close are you to covering a testing goal?
  - Smaller scores are typically better.
  - Must offer information to guide the search.
  - Must be cheap to calculate - performed 100s-1000s of times per generation.

Not Just Test Generation...

Metaheuristic search can be applied to any problem with:

- A large search space.
- Fitness function and solution generation methods with low computational complexity.
- Approximate continuity in the fitness function.
- No known optimal solution.

Writing code that writes code that ~~breaks~~ code.
fixes

Automated Program Repair

- Popular projects may have hundreds of bugs reported *per day*.
- Repair techniques, like GenProg, automatically produce patches that can repair common bug types.
- Many bugs can be fixed with just a few changes to the source code - inserting new code, deleting or moving existing code.
- GenProg uses the same ideas to *search* for repairs automatically.

GenProg

- **Genetic programming** - solutions represent sequences of edits to the source code.
- Each candidate patch is applied to the program to produce a new program.
- See if the patched program passes all tests.
  - Fitness function: how many tests pass?
- Use crossover and mutation to evolve better patches.

GenProg Results

- GenProg repaired 55 out of 105 bugs at an average cost of $8 per bug.
    - Large projects - over 5 million lines of code, 10000 test cases.
- Able to patch infinite loops, segmentation faults, buffer overflows, denial of service vulnerabilities, “wrong output” faults, and more.

Automated Code Transplantation

- Not just patches…
- Many coding tasks involve “reinventing the wheel” - redesigning and writing code to perform a function that already exists in some other project.
- What if we could slice out that code (“organ”) from a “donor” program and transplant it to the right “vein” in the target software?

muScalpel

- Uses a form of genetic programming.
- Initial population of 1 statement patches.
    - Organs need very few statements from the donor.
    - Starting with one line at a time allows muScalpel to find efficient solutions quickly.
- Search evolves both organs and veins.
    - Optimize the set of code transplanted from the donor, and the optimal location to place that code in the target software.
- Apply tests to ensure correctness of both original code and new features.

muScalpel Results

- Transplantation of H.264 video codec from x264 system to VLC media player.
  - Took VLC developers 20 days to write the code manually.
  - Took muScalpal 26 hours to transplant automatically.
- In 12 of 15 experiments, successful transplants that passed all tests.

We still need good engineers.

Are you that engineer?

Questions and Discussion

Interested in discussing more?

E-Mail: greg@greggay.com

Web: http://www.greggay.com

In-Person: 3A66 SWGN