When have we tested enough?

When to Stop Testing

Today’s Topics

- How do we know when we are done?
- Stopping Criteria
  - Coverage
  - Budget
  - Plan
  - Reliability
  - Mutation analysis

When do we stop?

The all important question

When have we tested enough?

When We Have Achieved Coverage

- Set your sights on some coverage criteria and test until that is achieved.

- Problems?

The Budget Coverage Criterion

- Industry’s answer to “when is testing done”
  - When the money is used up
  - When the deadline is reached

- Problems?

Plan to Test—Test to the Plan

- Plan your tests carefully; then test according to plan
- When the tests are done—you are done.
- Problems?

Categorizing and specifying the reliability of software systems

Software Reliability

Courtesy Prof. Mats Heimdahl

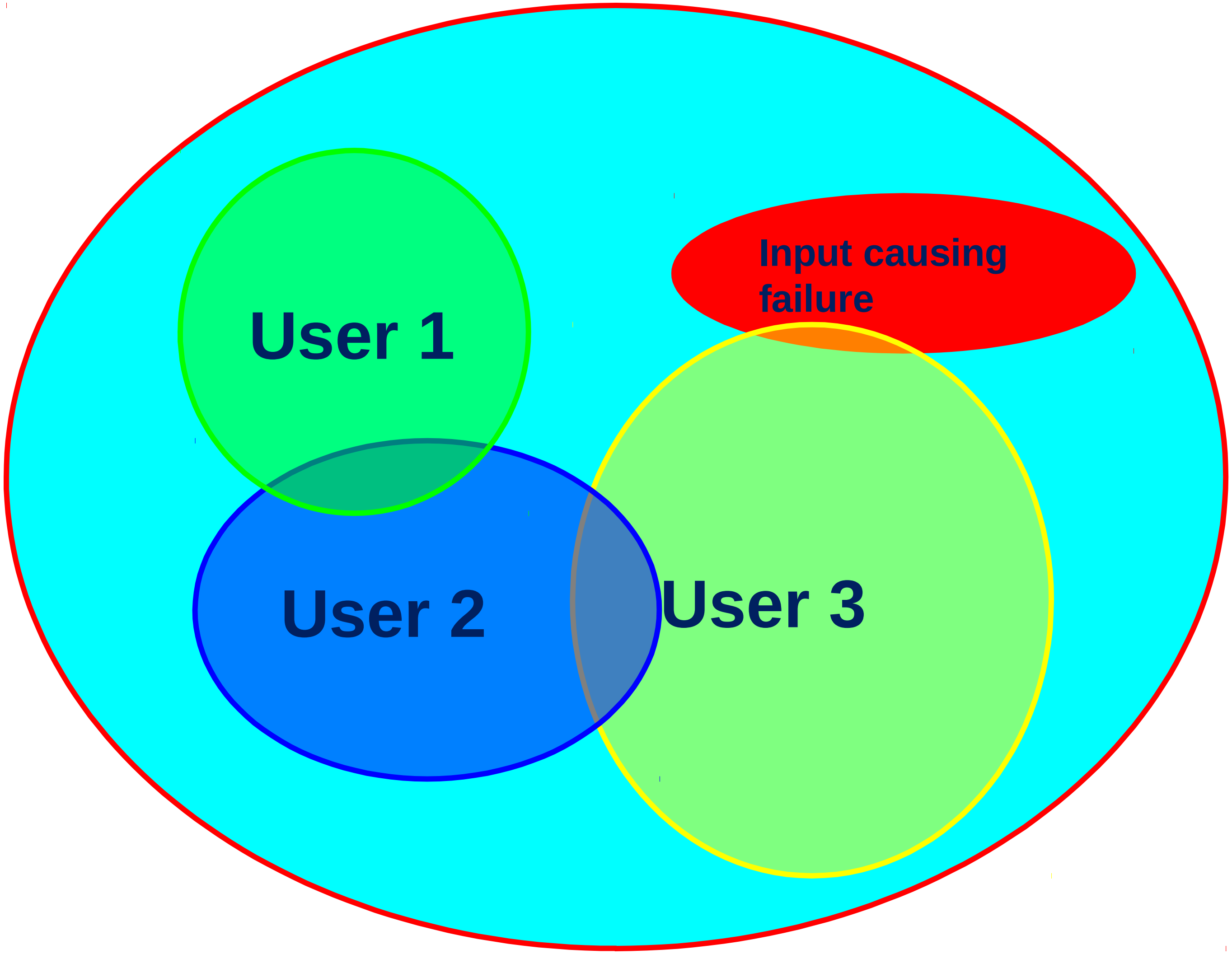
What Is Reliability?

- Probability of failure-free operation for a specified time in a specified environment for a given purpose
- This means quite different things depending on the system and the users of that system
- Informally, reliability is a measure of how well system users think it provides the services they require

Reliability Improvement

- Reliability is improved when software faults which occur in the most frequently used parts of the software are removed
- Removing x% of software faults will not necessarily lead to an x% reliability improvement
  - In a study, removing 60% of software defects actually led to a 3% reliability improvement
- Removing faults with serious consequences is the most important objective

Reliability Perception



Software Reliability

- Cannot be defined objectively
  - Reliability measurements which are quoted out of context are not meaningful
- Requires operational profile for its definition
  - The operational profile defines the expected pattern of software usage
- Must consider fault consequences
  - Not all faults are equally serious
  - System is perceived as more unreliable if there are more serious faults

Reliability and Efficiency

- Reliability is usually more important than efficiency
- No need to use hardware to fullest extent as computers are cheap and fast
- Unreliable software is not used
- Hard to improve unreliable systems
- Software failure costs often far exceed system costs
- Costs of data loss are very high

Reliability Metrics

- Hardware metrics not really suitable for software as they are based on component failures and the need to repair or replace a component once it has failed
  - The design is assumed to be correct
- Software failures are always design failures
  - Often the system continues to be available in spite of the fact that a failure has occurred

Reliability Metrics

- Probability of failure on demand
  - This is a measure of the likelihood that the system will fail when a service request is made
  - POFOD = 0.001 means 1 out of 1000 service requests result in failure
- Rate of fault occurrence (ROCOF)
  - Frequency of occurrence of unexpected behavior
  - ROCOF of 0.02 means 2 failures are likely in each 100 operational time units

Reliability Metrics

- Mean time to failure
  - Measure of the time between observed failures
  - MTTF of 500 means that the time between failures is 500 time units
- Availability
  - Measure of how likely the system is available for use. Takes repair/restart time into account
  - Availability of 0.998 means software is available for 998 out of 1000 time units

Reliability Examples

- Provide software with 10,000 inputs
  - Wrong result on 35, Crash on 5
  - What is the POFOD?
- Run the software for 144 hours (6*106 inputs)
  - Software failed on 6 input
  - What is the ROCOF?
  - What is the POFOD?
- You you a piece of software with the advertised ROCOF of 0.001 failures/hour for “stop failures”
  - You know it takes 3 hours (on average) to get the system up again after a failure
  - What is the availability?

Reliability Examples

- Provide software with 10,000 inputs
  - Wrong result on 35, Crash on 5
  - What is the POFOD?

$$\longrightarrow \frac{40}{10,000} \longrightarrow 0.004$$

- Run the software for 144 hours (6*10^6 inputs)
  - Software failed on 6 input
  - What is the ROCOF?
  - What is the POFOD?
- You have a piece of software with the advertised ROCOF of 0.001 failures/hour for “stop failures”
  - You know it takes 3 hours (on average) to get the system up again after a failure
  - What is the availability

Reliability Examples

- Provide software with 10,000 inputs
  - Wrong result on 35, Crash on 5
  - What is the POFOD?

$$\frac{40}{10,000} \longrightarrow 0.004$$

- Run the software for 144 hours (6*10^6 inputs)
  - Software failed on 6 input
  - What is the ROCOF?

$$\frac{6}{144} \longrightarrow 0.04 \longrightarrow \frac{1}{24}$$

  - What is the POFOD?

$$\frac{6}{6*10^{6}} \longrightarrow 10^{-6}$$

- You have a piece of software with the advertised ROCOF of 0.001 failures/hour for “stop failures”
  - You know it takes 3 hours (on average) to get the system up again after a failure
  - What is the availability

Reliability Examples

- Provide software with 10,000 inputs
  - Wrong result on 35, Crash on 5 → $\frac{40}{10,000} \rightarrow$ **0.004**
  - What is the POFOD?
- Run the software for 144 hours (6*10^6 inputs)
  - Software failed on 6 input → $\frac{6}{144} \rightarrow$ **0.04** → $\frac{1}{24}$
  - What is the ROCOF?
  - What is the POFOD? → $\frac{6}{6*10^6} \rightarrow$ **10^-6**
- You have a piece of software with the advertised ROCOF of 0.001 failures/hour for “stop failures”
  - You know it takes 3 hours (on average) to get the system up again after a failure
  - What is the availability → **0.997**

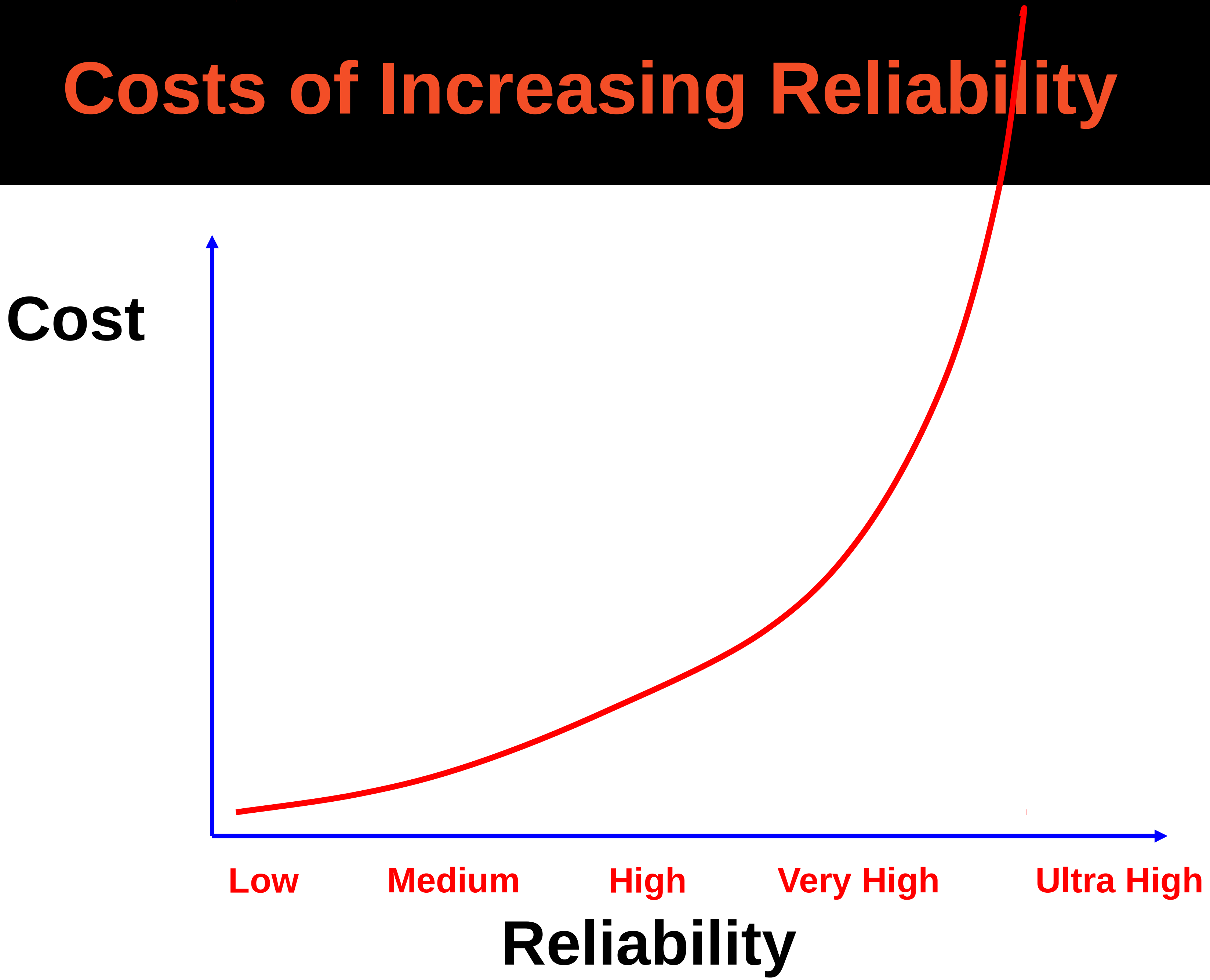
Reliability Measurement

- Measure the number of system failures for a given number of system inputs
  - Used to compute POFOD
- Measure the time (or number of transactions) between system failures
  - Used to compute ROCOF and MTTF
- Measure the time to restart after failure
  - Used to compute AVAIL

Reliability Economics

- Because of very high costs of reliability achievement, it may be more cost effective to accept unreliability and pay for failure costs
- However, this depends on social and political factors
    - A reputation for unreliable products may lose future business
- Depends on system type
    - For business systems in particular, modest reliability may be adequate

Costs of Increasing Reliability



Statistical Testing

- Testing software for reliability rather than fault detection
- Test data selection should follow the predicted usage profile for the software
- Measuring the number of errors allows the reliability of the software to be predicted
- An acceptable level of reliability should be specified and the software tested and amended until that level of reliability is reached

Statistical Testing Procedure

- Determine operational profile of the software
- Generate a set of test data corresponding to this profile
- Apply tests, measuring amount of execution time between each failure
- After a statistically valid number of tests have been executed, reliability can be measured

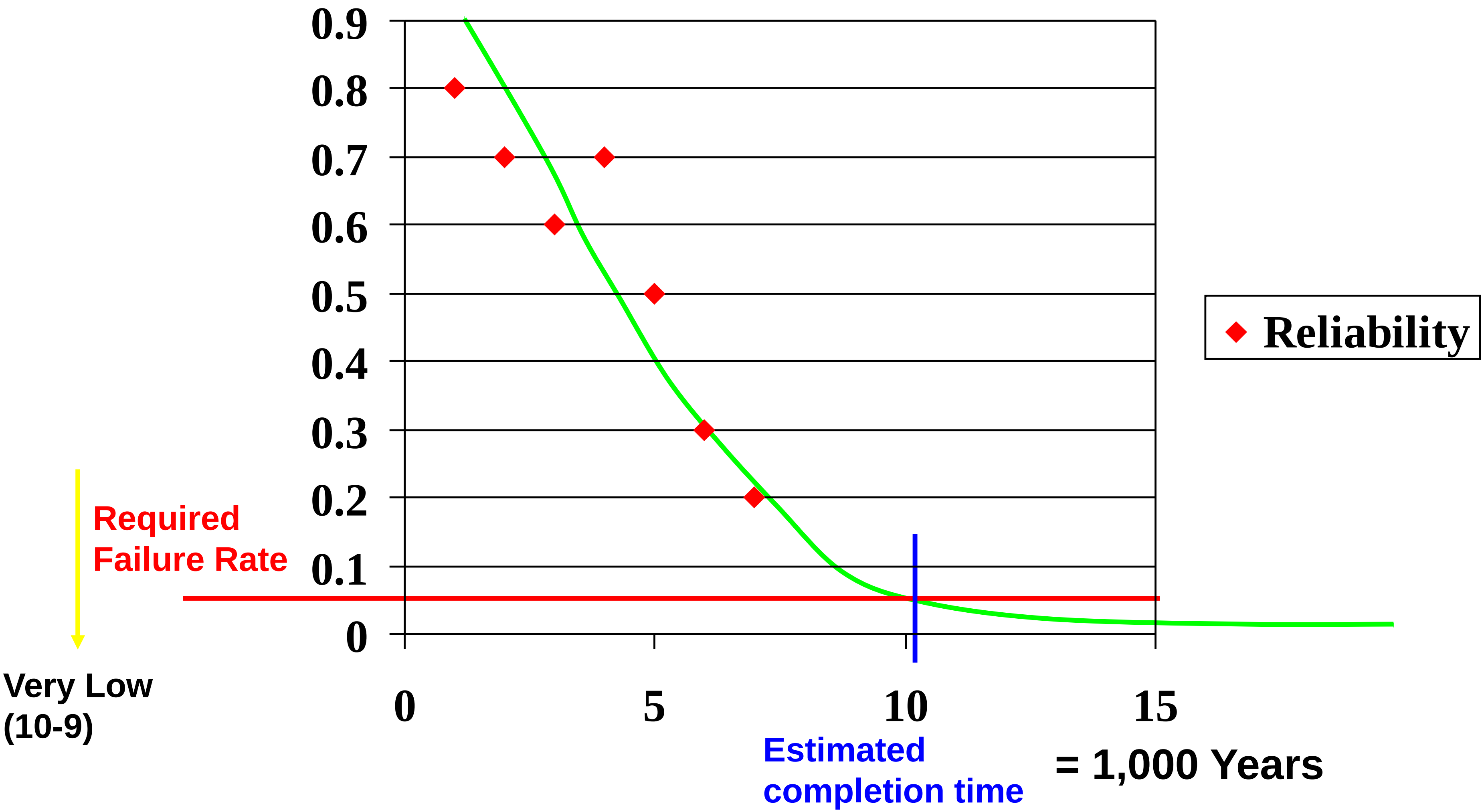
Statistical Testing Difficulties

- Uncertainty in the operational profile
  - This is a particular problem for new systems with no operational history
  - Less of a problem for replacement systems
- High costs of generating the operational profile
  - Costs are very dependent on what usage information is collected by the organization which requires the profile
- Statistical uncertainty when high reliability is specified
  - Difficult to estimate level of confidence in operational profile
  - Usage pattern of software may change with time

Reliability Growth Modeling

- Growth model is a mathematical model of the system reliability change as it is tested and faults are removed
- Used as a means of reliability prediction by extrapolating from current data
- Depends on the use of statistical testing to measure the reliability of a system version

Reliability Prediction



Key Points

- Reliability is usually the most important dynamic software characteristic
- Professionals should aim to produce reliable software
- Reliability depends on the pattern of usage of the software
  - Faulty software can be reliable
- Reliability requirements should be defined quantitatively whenever possible

Key Points

- There are many different reliability metrics
  - The metric chosen should reflect the type of system and the application domain
- Statistical testing is used for reliability assessment
  - Depends on using a test data set which reflects the use of the software
- Reliability growth models may be used to predict when a required level of reliability will be achieved

An approach to figuring out if the test set is any good

Mutation Testing

Courtesy Prof. Mats Heimdahl

Question

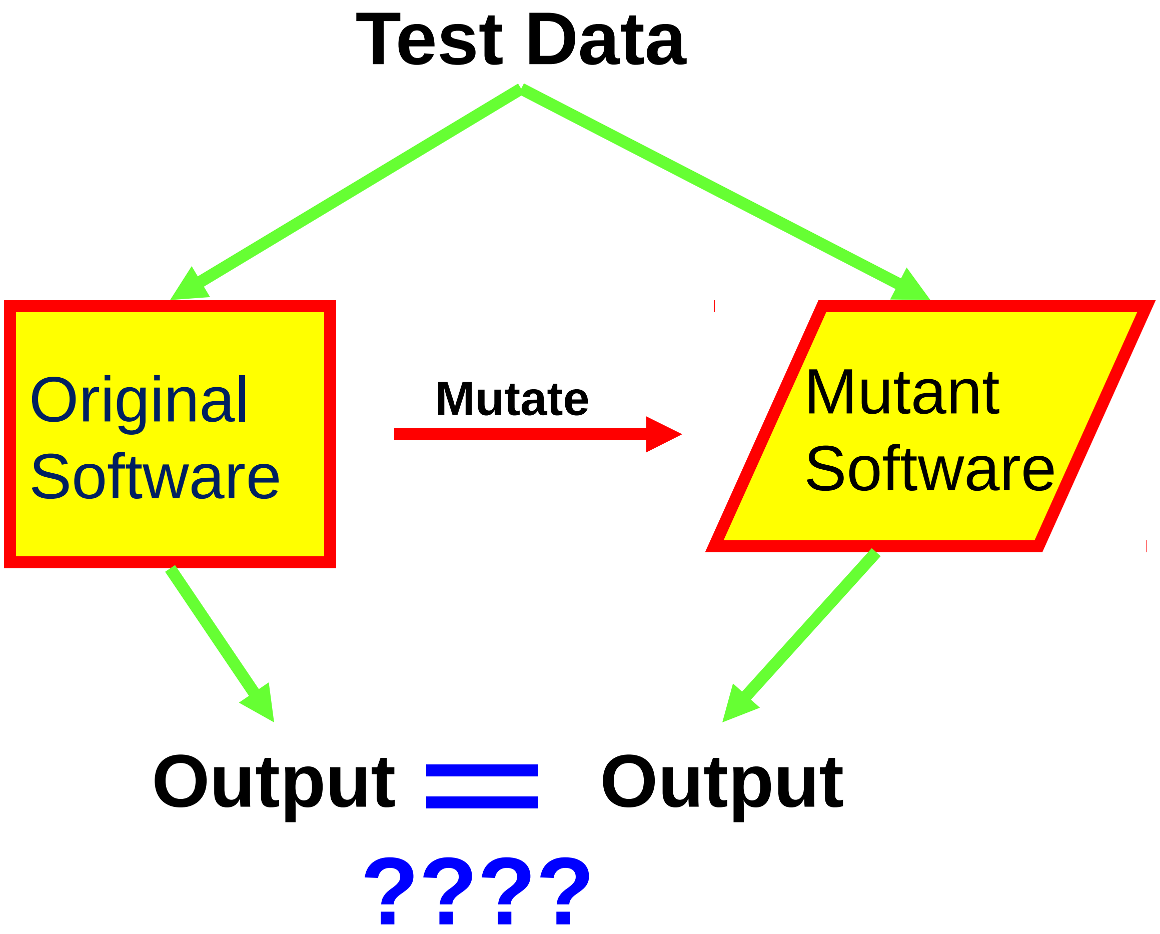
- I have a collection of test cases
- How do I know if the set is any good?
  - That is, how likely is it to reveal faults?

Mutation Testing

- An approach to investigating the quality of your test data

- Create a second version of your software with some minor change
  - Introduce a “mutation”
- Run the test cases and see if they reveal the mutation (an artificial fault)
  - If yes – Good test data
  - If no – Bad test data

General Idea



Test Data

Original Software

Mutate

Mutant Software

Output = Output

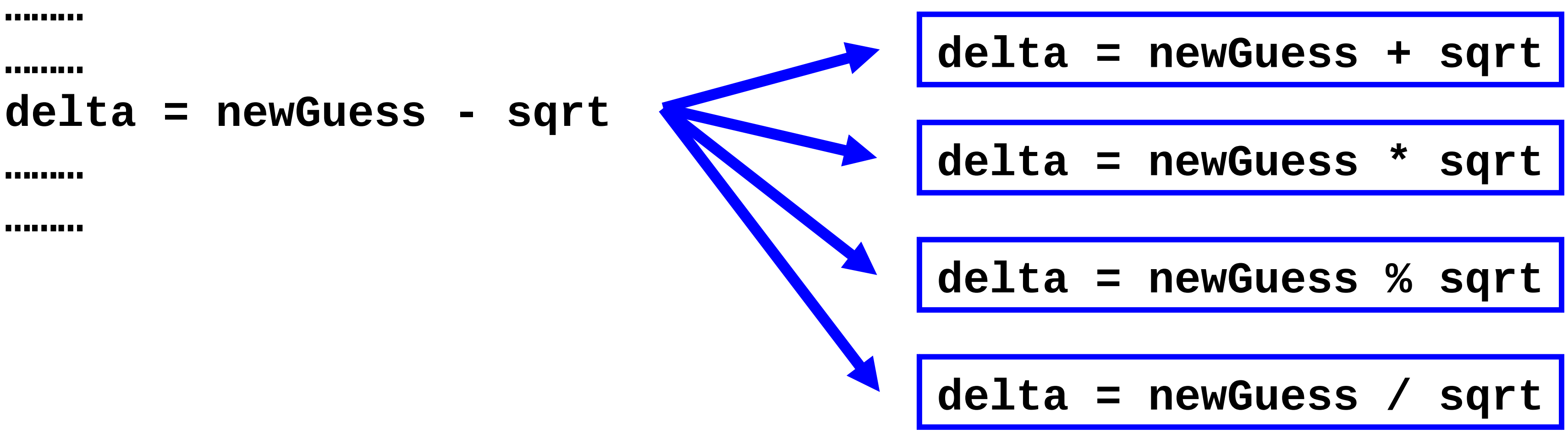
????

What is a Mutant?

- A mutant is the original program with a small change introduced
  - The change is called a mutation
- A mutation is one single “change” on one line in the original
  - The “change” is caused by a mutation operator
  - Also called mutagens, mutagenic operators, etc.
- For the program P, the set of mutants are called the neighborhood of P

How do we Create a Mutant?

- Apply appropriate mutation operators to each line in the program



```
………
………
delta = newGuess - sqrt
………
………
```

```
delta = newGuess + sqrt
```

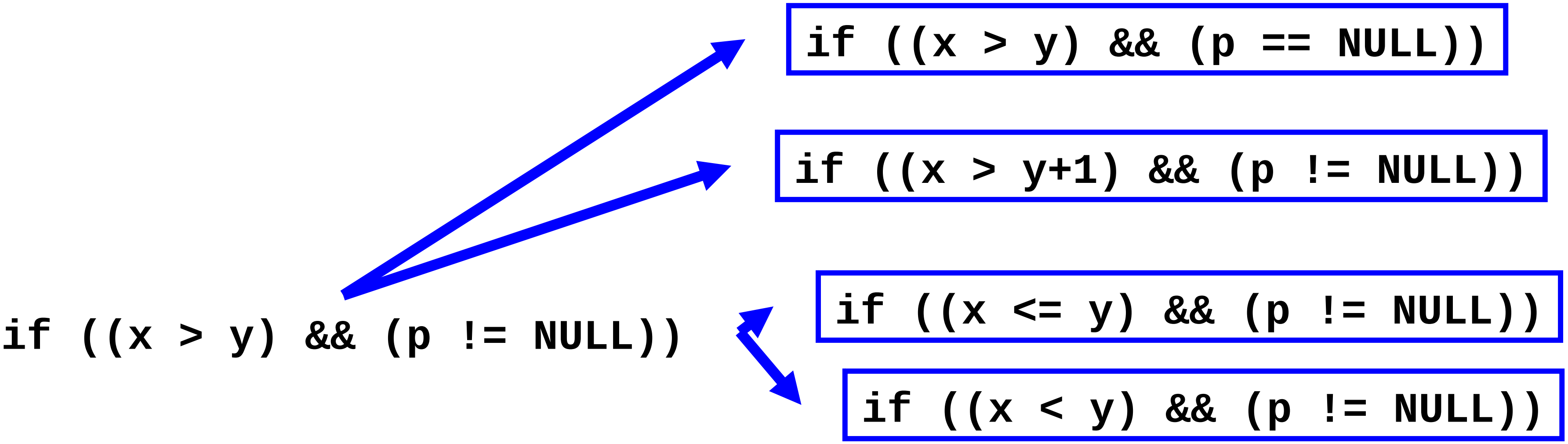
```
delta = newGuess * sqrt
```

```
delta = newGuess % sqrt
```

```
delta = newGuess / sqrt
```

Mutation Operators

- There are many different mutation operators
  - Operators
  - Off by one
  - Switch variable names of same type



```
if ((x > y) && (p != NULL))
```

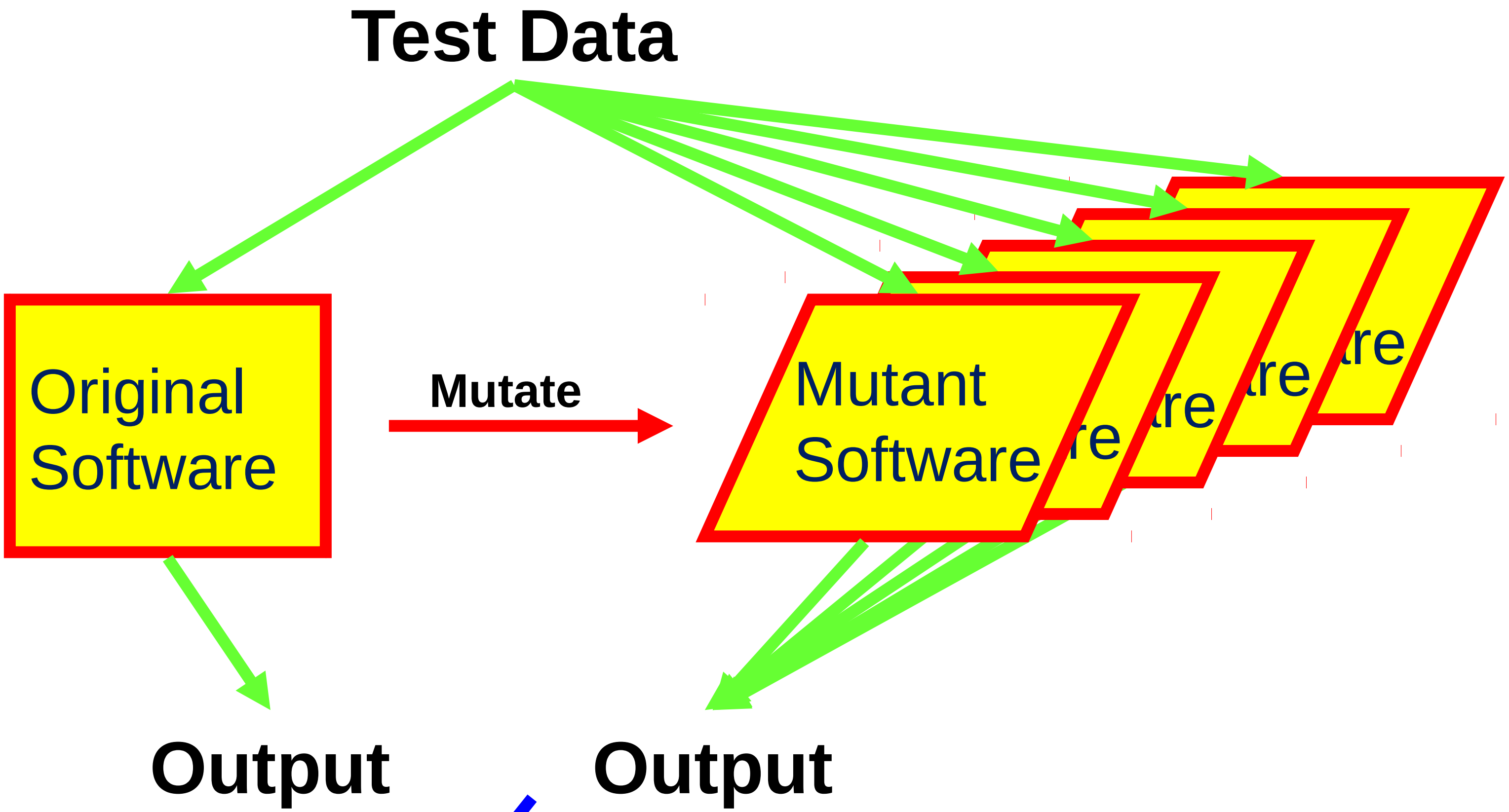
```
if ((x > y) && (p == NULL))
```

```
if ((x > y+1) && (p != NULL))
```

```
if ((x <= y) && (p != NULL))
```

```
if ((x < y) && (p != NULL))
```

Testing Approach



Test Data

Original Software

Mutate

Mutant Software

Output

Output

≠ Kill Mutant!

= Problem!

Mutation Adequacy Score (MS)

- How well did you do?

$$\text{MS} = \frac{\#\ \text{Dead}}{\#\ \text{Mutants} - \#\ \text{Equivalent}} * 100\ \%$$

- What are the problems of this approach?

Test Adequacy Summary

How do we know if our tests are any good?

- Code coverage criteria
  - Hard to achieve
  - Experiments indicate they are no better (or marginally better) than random testing
- Use the test plan

- Statistical
  - User profile or input distribution
- Mutation testing

Six Essentials of Testing

- The quality of the test process determines the success of the test effort
- Prevent defect migration by using early life-cycle testing techniques
- The time for software testing tools is now

Adapted from Software Testing in the Real World, Edward Kit; Addison-Wesley, 1995

Six Essentials of Testing

- A real person must take responsibility for improving the testing process
- Testing is a professional discipline requiring trained, skilled people
- Cultivate a positive team attitude of creative destruction

Adapted from Software Testing in the Real World, Edward Kit; Addison-Wesley, 1995