SE 333/433 Software Testing & Quality Assurance

Introduction :

Software Quality and Software Testing

Outline

- Introductions
- Syllabus
- Software Quality Assurance: Introduction
  - Software Quality
  - Software Testing
- Road map

Overview

- What is software quality?

- How can it be measured?
  - How can it be measured before/after the software is delivered?

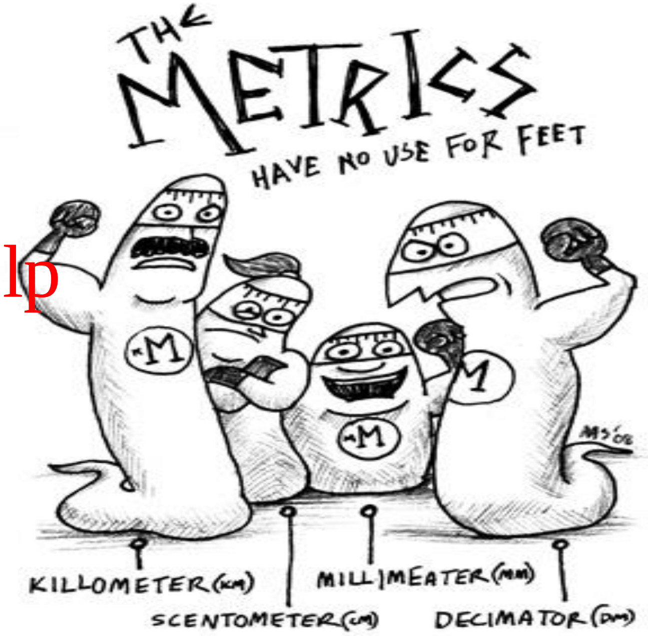
Quality ?

- Think of an everyday object
  - e.g. a chair
  - How would you measure it's "quality"?
    - construction quality? (e.g. strength of the joints,…)
    - aesthetic value? (e.g. elegance,…)
    - fit for purpose? (e.g. comfortable,…)

- All quality measures are relative
  - there is no absolute scale
  - we can say A is better than B but it is usually hard to say how much better

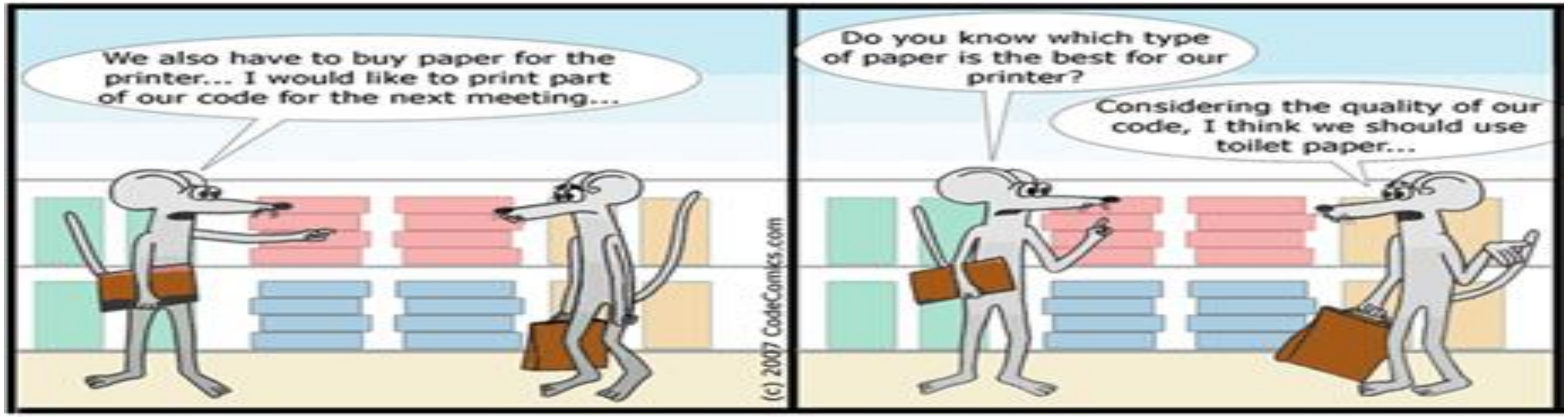
Examples of Metrics from Everyday Life

- Working and living
  - Cost of utilities for the month
  - Cost of groceries for the month
  - Amount of monthly rent per month
  - Time spent at work each Saturday for the past month
  - Time spent mowing the lawn for the past two times
- College experience
  - Grades received in class last quarter
  - Number of classes taken each quarter
  - Amount of time spent in class this week
  - Amount of time spent on studying and homework this week
  - Number of hours of sleep last night
- Travel
  - Time to drive from home to the airport
  - Amount of miles traveled today
  - Cost of meals and lodging for yesterday



What is Software Quality ?

- Conformance to requirements.
- Narrowest sense of software quality.
  - Lack of bugs.
  - High reliability (number of failures per *n* hours of operation).


We also have to buy paper for the printer... I would like to print part of our code for the next meeting...
(c) 2007 CodeComics.com
Do you know which type of paper is the best for our printer?
Considering the quality of our code, I think we should use toilet paper...

What is Software Quality ?

- According to the IEEE, Software quality is:
  1. The degree to which a system, component, or process meets specified requirements.
  2. The degree to which a system, component, or process meets customer or user needs or expectations.

Software Quality :

- Definition:

  Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software

- Three important points in this definition
  - Explicit software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality
  - Specific standards define a set of development criteria that guide the manner in which software is engineered.
  - There is a set of implicit requirements that often goes unmentioned (e.g., ease of use). If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect

DEPAUL UNIVERSITY

ISO 9126 Software Quality Factors

- Functionality
  - The degree to which the software satisfies stated needs
- Reliability
  - The amount of time that the software is available for use
- Usability
  - The degree to which the software is easy to use
- Efficiency
  - The degree to which the software makes optimal use of system resources
- Maintainability
  - The ease with which repair and enhancement may be made to the software
- Portability
  - The ease with which the software can be transposed from one environment to another

Key Quality Concepts

- Reliability
  - designer must be able to predict how the system will behave:
    - **completeness** - does it do everything it is supposed to do? (e.g. handle all possible inputs)
    - **consistency** - does it always behave as expected? (e.g. repeatability)
    - **robustness** - does it behave well under abnormal conditions? (e.g. resource failure)
- Efficiency
  - Use of resources such as processor time, memory, network bandwidth

DePaul University

Key Quality Concepts

- Maintainability
  - How easy will it be to modify in the future?
  - perfective, adaptive, corrective

- Usability
  - How easy is it to use?

How is Software Quality is measured?

- Metric:
  - (IEEE) A quantitative measure of the degree to which a system, component, or process possesses a given attribute
- Purpose
  - Aid in the evaluation of analysis and design models
  - Provide an indication of the complexity of procedural designs and source code

DePaul University

Metrics for Object Oriented Design

- Number of children (i.e., subclasses)
  - As the number of children of a class grows
    - Reuse increases
    - The abstraction represented by the parent class can be diluted by inappropriate children
    - The amount of testing required will increase

DePaul University

Metrics for Object Oriented Design

- Coupling between object classes
  - Measures the number of collaborations a class has with any other classes
  - Higher coupling decreases the reusability of a class
  - Higher coupling complicates modifications and testing
  - Coupling should be kept as low as possible

Comment Percentage (CP)

- Number of commented lines of code divided by the number of non-blank lines of code

- Usually 20% indicates adequate commenting for C or Java code
- The higher the CP value the more maintainable the module is

Outline

- Introductions
- Syllabus
- Software Quality Assurance: Introduction
  - Software Quality
  - Software Testing
- Road map

Software Testing: Overview

- **Software Testing?**
- **Why Test?**
- **What Do We Do When We Test ?**
  - Understand basic techniques for software verification and validation
  - Analyze basics of software testing techniques

A Concrete Example…

Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the number of occurrences of 0 in arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
   {
      if (arr [ i ] == 0)
      {
         count++;
      }
   }
   return count;
}
```

Test 1

[2, 7, 0]

Expected: 1

Actual: 1

Error : i is 1, not 0, on the first iteration

Symptoms: none

Test 2

[0, 2, 7]

Expected: 1

Actual: 0

Error: i is 1, not 0

Error propagates to the variable count

Symptoms: count is 0 at the return statement

Myth Busters

Software Testing

Myth #1 in Software Testing

Q: What is the objective of software testing?

A: Testing is to show that there are no errors/bugs/defects in the software.

- Fact:
  - No!! The main objective of testing is to *discover* defects.
  - Testing is a *destructive* activity.

Myth #2 in Software Testing

Q: What is the objective of software testing?

A: Testing is to ensure that the software does what it is supposed to do.

- Fact:
  - Only partly true.
  - Testing is also to ensure the software *does not* do what it is *not supposed* to do.

Myth #3 in Software Testing

Q: How challenging is software testing?

A: Testing is easier than design and implementation.

- Fact:
  - Must consider all possible scenarios.
  - Implied and unstated requirements and threats.
  - Must be imaginative and creative.

Myth #4 in Software Testing

Q: How challenging is software testing?

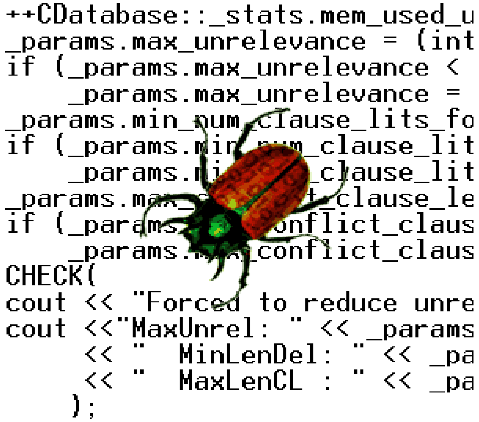
A: Testing is an extremely creative and intellectually challenging task.

The Term Bug

- Bug is used informally

- Defect
- Fault
- Problem
- Error
- Incident
- Anomaly
- Variance

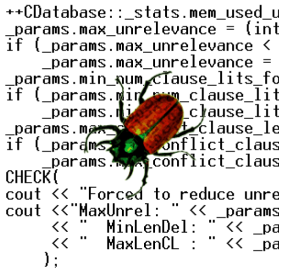
- Failure
- Inconsistency
- Product Anomaly
- Product Incidence
- Feature



```
++CDatabase::_stats.mem_used_u
_params.max_unrelevance = (int
if (_params.max_unrelevance <
    _params.max_unrelevance =
_params.min_num_clause_lits_fo
if (_params.min_num_clause_lit
    _params.min_num_clause_lit
_params.max_conflict_clause_le
if (_params.max_conflict_claus
    _params.max_conflict_claus
CHECK(
cout << "Forced to reduce unre
cout <<"MaxUnrel: " << _params
     << "  MinLenDel: " << _pa
     << "  MaxLenCL : " << _pa
    );
```

Failures

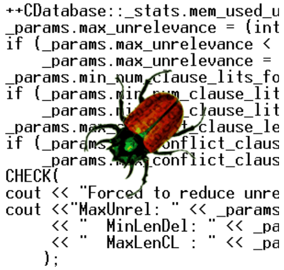
- ***Failures*** are
  - deviation of the observed behavior of a system from its specification, i.e., its expected behavior.
- Failures can only be determined with respect to the specifications.
- Failures are concerned with the observed behavior and outcome of the system.



```
++CDatabase::_stats.mem_used_u
_params.max_unrelevance = (int
if (_params.max_unrelevance <
    _params.max_unrelevance =
_params.min_num_clause_lits_fo
if (_params.min_num_clause_lit
    _params.min_num_clause_lit
_params.max_conflict_clause_le
if (_params.max_conflict_claus
    _params.max_conflict_claus
CHECK(
cout << "Forced to reduce unre
cout <<"MaxUnrel: " << _params
     << "  MinLenDel: " << _pa
     << "  MaxLenCL : " << _pa
    );
```

Defects

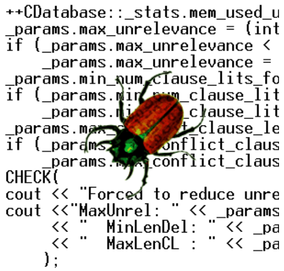
- ***Defects*** are
  - flaws in a system that can cause the system to fail to perform its required function
    - e.g. an incorrect condition or statement.
- Defects are concerned with specific parts or components of the system.
- Defects are synonymous with *faults*



```
++CDatabase::_stats.mem_used_u
_params.max_unrelevance = (int
if (_params.max_unrelevance <
    _params.max_unrelevance =
_params.min_num_clause_lits_fo
if (_params.min_num_clause_lit
    _params.mi[illegible]_clause_lit
_params.ma[illegible]t_clause_le
if (_params.[illegible]conflict_claus
    _params.max_conflict_claus
CHECK(
cout << "Forced to reduce unre
cout <<"MaxUnrel: " << _params
     << "  MinLenDel: " << _pa
     << "  MaxLenCL : " << _pa
    );
```

Errors

- *Errors* are
  - human actions that result in a fault or defect in the system.
- Errors are concerned with the underlying causes of the defects.
- Errors are synonymous with *mistakes.*



```
++CDatabase::_stats.mem_used_u
_params.max_unrelevance = (int
if (_params.max_unrelevance <
    _params.max_unrelevance =
_params.min_num_clause_lits_fo
if (_params.min_num_clause_lit
    _params.mi[illegible]_clause_lit
_params.max_[illegible]t_clause_le
if (_params.[illegible]onflict_claus
    _params.max_conflict_claus
CHECK(
cout << "Forced to reduce unre
cout <<"MaxUnrel: " << _params
     << "  MinLenDel: " << _pa
     << "  MaxLenCL : " << _pa
    );
```

The Relations among Failures, Defects, and Errors

- A human being makes an *error* (*mistake*)
  - can occur in design, coding, requirements, even testing.
- An *error* can lead to a *defect* (*fault*)
  - can occur in requirements, design, or program code.
- If a *defect* in code is executed, a *failure* may occur.
  - Failures only occur when a *defect* in the code is executed.
  - Not all defects cause failures all the time.
- Defects occur because human beings are fallible
- Failures can be caused by environmental conditions as well.

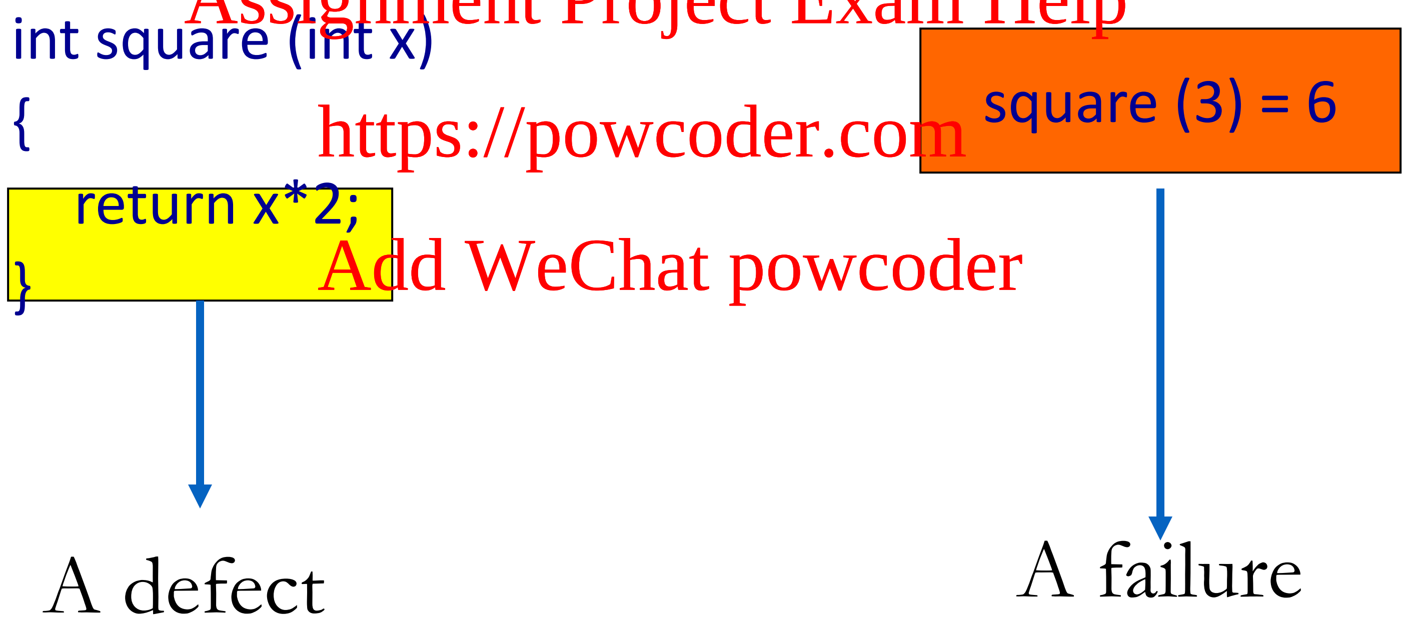
DEPAUL UNIVERSITY

The Relations among Failures, Defects, and Errors

- The terms error, failure and defect have different meaning when testing. Especially in using JUnit. In this case:
- Test Case Verdicts
  - Pass
    - The test case execution was completed.
    - The function being tested performed as expected
  - Fail
    - The test case execution was completed
    - The function being tested did not perform as expected
  - Error
    - The test case execution was not completed, due to an unexpected event, exceptions, or improper set up of the test case, etc.

Failures vs. Defects: A Simple Example

- For any integer n, square (n) = n*n.



```
int square (int x)
{
    return x*2;
}
```

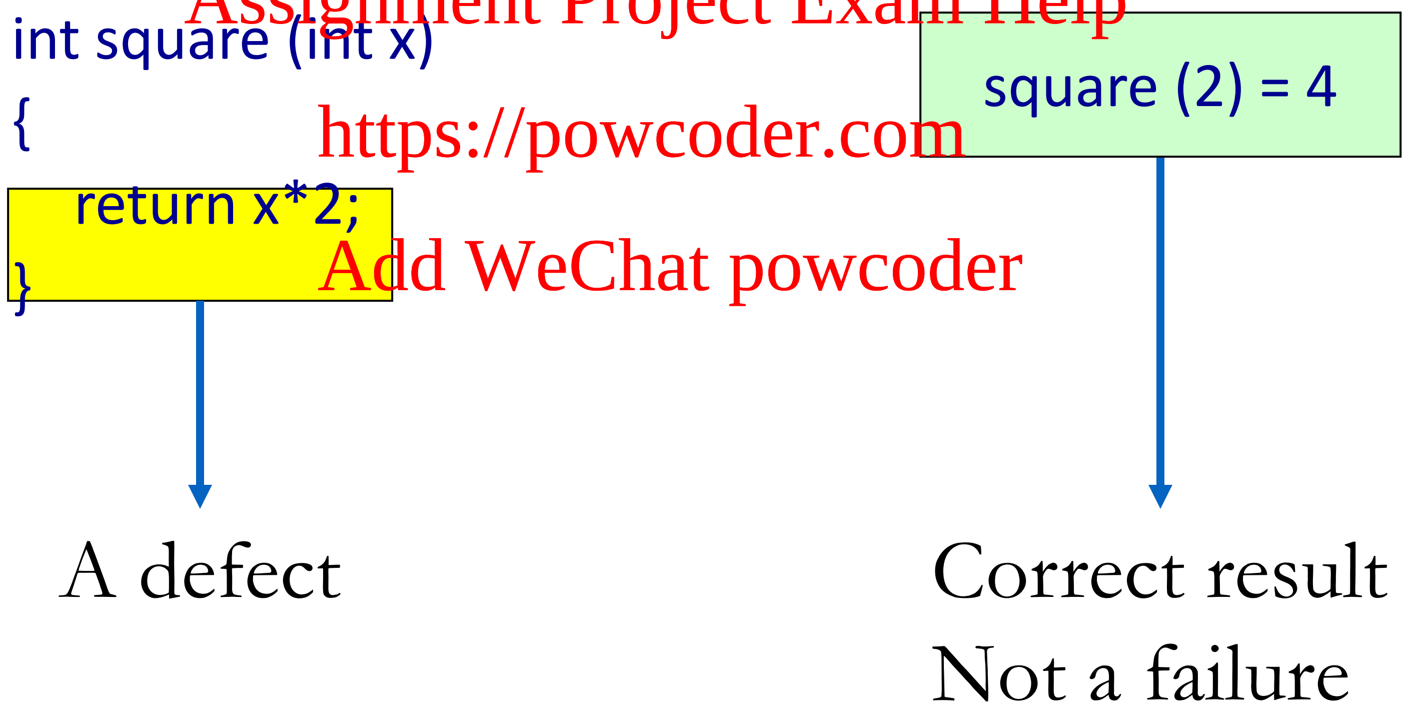
square (3) = 6

A defect

A failure

Failures vs. Defects: A Simple Example

- For any integer n, square (n) = n*n.



```
int square (int x)
{
    return x*2;
}
```

square (2) = 4

A defect

Correct result
Not a failure

Software Testing

- ***Software testing*** is
  - the process of *executing a program* (or parts of a program) with the intention of finding defects
- The purpose of testing
  - to find defects.
  - to discover every conceivable weakness in a software product.

1. Software testing $\neq$ Debugging.
2. Software testing $\neq$ Quality assurance

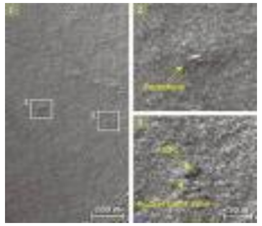
Software Testing vs. Quality Assurance (QA)

- Testing is necessary, but not sufficient for quality assurance
  - Testing contributes to improve quality by identifying problems.
- Quality assurance sets the standards for the team/organization to build better software.

Software is a Skin that Surrounds Our Civilization

Spectacular Software Failures

- NASA’s Mars lander: September 1999, crashed due to a units integration fault



- Ariane 5 explosion : Very expensive

Ariane 5: exception-handling bug : forced self destruct on maiden flight (64-bit to 16-bit conversion: about 370 million $ lost)

Northeast Blackout of 2003

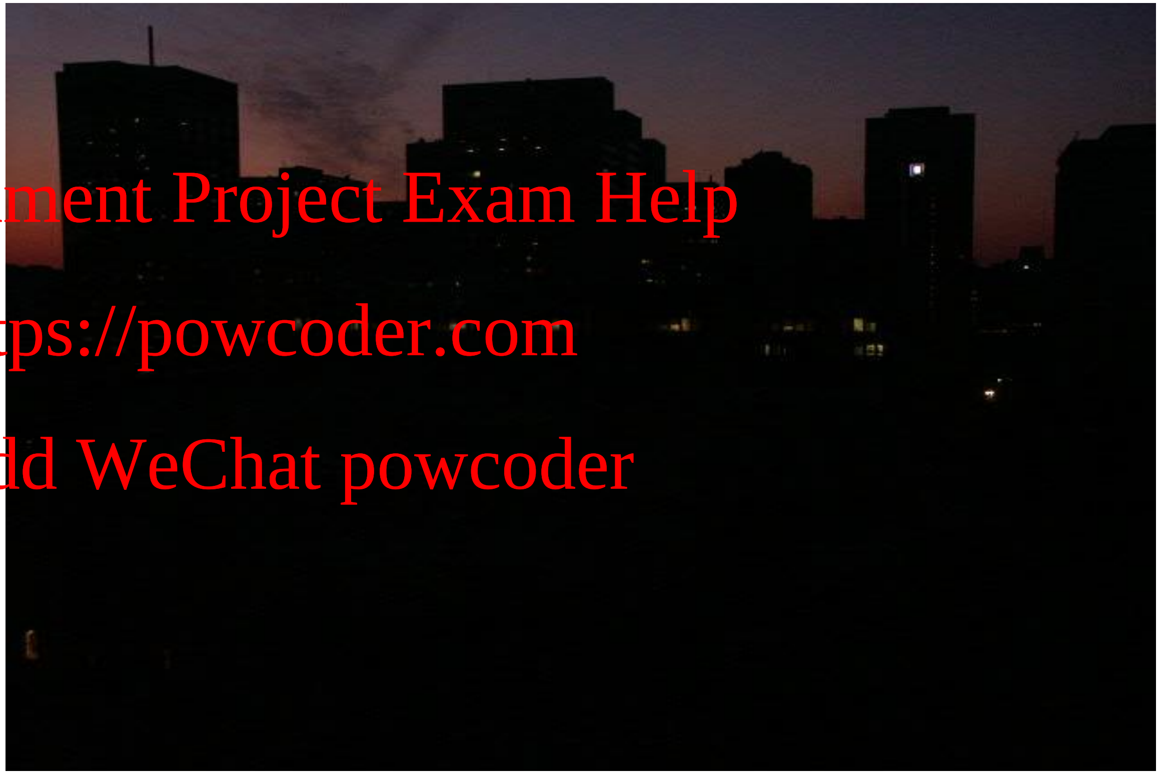
508 generating units and 256 power plants shut down

Affected 10 million people in Ontario, Canada

Affected 40 million people in 8 US states

Financial losses of

$6 Billion USD



The alarm system in the energy management system failed due to a software error and operators were not informed of the power overload in the system

Costly Software Failures !

- NIST report, “The Economic Impacts of Inadequate Infrastructure for Software Testing” (2002)
  - ➢ Inadequate software testing costs the US alone between $22 and $59 billion annually

- Huge losses due to web application failures
  - ➢ Financial services : $6.5 million per hour (just in USA!)
  - ➢ Credit card sales applications : $2.4 million per hour (in USA)

Discussion …

- Have you heard of other software bugs?
  - In the media?
  - From personal experience?
- Does this embarrass you as a (future) software engineer?

Cost of Not Testing

Poor Program Managers might say:
"Testing is too expensive."

- Testing is the **most time consuming and expensive part of software development**
- Not testing is even **more expensive**
- If we do not have enough testing effort early, the cost of testing **increases**

Testing Goals

- **The Major Objectives of Software Testing:**

  - Detect errors (or bugs) as much as possible in a given timeline.

  - Demonstrate a given software product matching its requirement specifications.

  - Validate the quality of a software testing using the minimum cost and efforts.

- **Testing can NOT prove product works 100%**

Testing Overview

- **Who tests**
  - ***Programmers***
  - ***Testers/Req. Analyst***
  - ***Users***
- **What is tested**
  - **Unit Code** testing
  - **Functional Code** testing
  - Integration/**system** testing
  - **User interface** testing

- **How (test cases designed)**
  - – **Intuition**
  - – **Specification based (*black box)***
  - – **Code based (*white-box)***

Exhaustive Testing is Hard

```
int max(int x, int y)
{
  if (x > y)
    return x;
  else
    return x;
}
```

18446744073709551616 possibilities

- Number of possible test cases (assuming 32 bit integers)
  - $2^{32} \times 2^{32} = 2^{64}$
- Do bigger test sets help?
  - **Test set {(x=3,y=2), (x=2,y=3)} will detect the error**
  - **Test set {(x=3,y=2),(x=4,y=3),(x=5,y=1)} will not detect the error although it has more test cases**
- It is not the number of test cases
- But, if $T_1 \supseteq T_2$, then T_1 will detect every fault detected by T_2

Exhaustive Testing is Hard

- Assume that the input for the `max` procedure was an integer array of size *n*
  - Number of test cases: $2^{32\times n}$

- Assume that the size of the input array is not bounded
  - Number of test cases: ∞

Generating Test Cases Randomly

```
bool isEqual(int x, int y)
{
  if (x = y)
    z := false;
  else
    z := false;
  return z;
}
```

0.00000000023283064365386962890625

- If we pick test cases randomly it is unlikely that we will pick a case where x and y have the same value
- If x and y can take 2^{32} different values, there are 2^{64} possible test cases. In 2^{32} of them x and y are equal
- **probability of picking a case where x is equal to y is 2^{-32}**
- It is not a good idea to pick the test cases randomly (with uniform distribution) in this case
- So, naive random testing is pretty hopeless too

Mutation Testing

1. Induce small changes to the program: mutants
2. Find tests that cause the mutant programs to fail: killing mutants
3. Failure is defined as different output from the original program
4. Check the output of useful tests on the original program

- Example program and mutants

```
if (x > y)
    z = x - y;
else
   z = 2 * x;
```

```
if (x > y)
∆if (x >= y)
    z = x - y;
    ∆ z = x + y;
    ∆ z = x – m;
else
     z = 2 * x;
```

Types of Testing

- **Unit (Module) testing**
  - testing of a single module in an isolated environment
- **Integration testing**
  - testing parts of the system by combining the modules
- **System testing**
  - testing of the system as a whole after the integration phase
- **Acceptance testing**
  - testing the system as a whole to find out if it satisfies the requirements specifications

Information

Project Proposal :

I will create empty groups in D2L where you can self-enroll.

I will create a project discussion for those who are looking for members, or want to join a group can post it there

Choose your partner(s) for the project and enroll in an empty group in D2L

Outline

- Introductions
- Syllabus
- Software Quality Assurance: Introduction
  - Software Quality
  - Software Testing
- Road map