Introduction to Software Testing

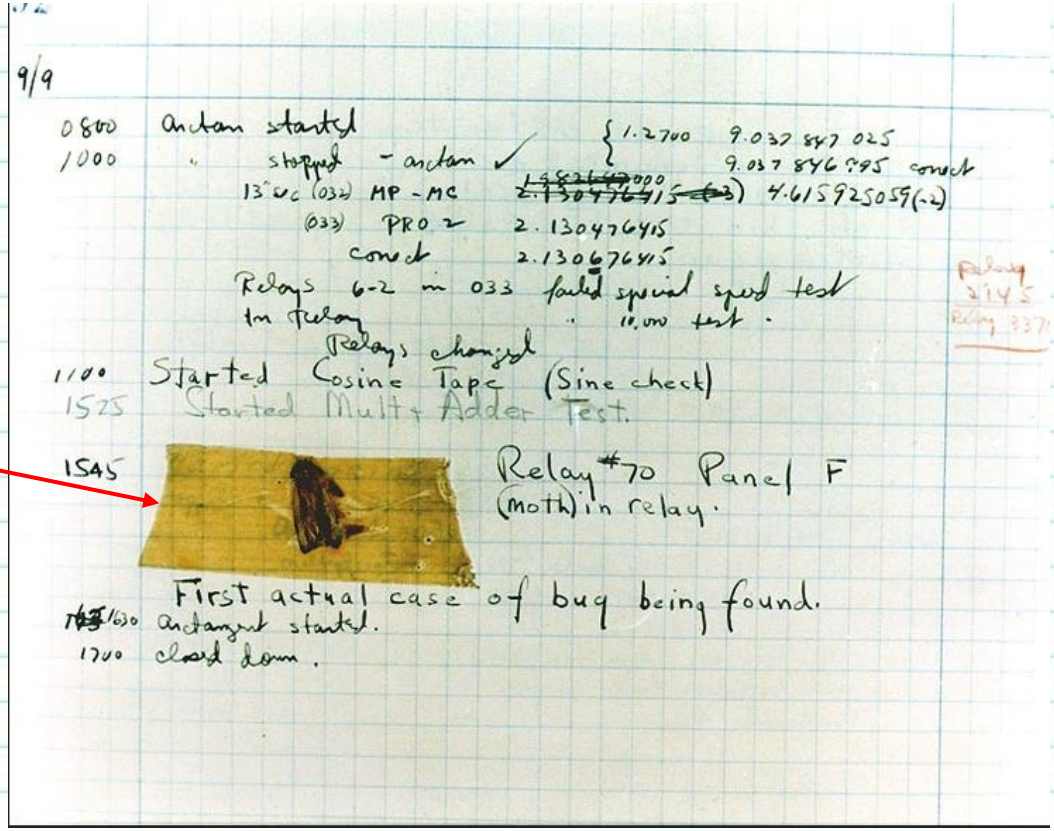
Dr. John H Robb, PMP, IEEE SEMC
UTA Computer Science and Engineering

Software is Ubiquitous

- Everywhere!
  - Automotive, phones, industrial control, power distribution, embedded devices, aerospace, military, IT systems
- Software is used in high security and/or safety applications
  - Medical devices, nuclear reactors, aerospace (commercial and military)
  - Banking Systems, identification systems, surveillance systems
- The expectation that software will work reliably places important emphasis on software test
- Robb's axiom: the software is only as good as the test function (or effort)

Where Do Bugs Come From?

- Well they started here...



The first computer bug!

- Steven M. Bellovin, a computer science professor at Columbia University. “most security problems are due to buggy code.”
- Spectacular software failures described here: http://en.wikipedia.org/wiki/List_of_software_bugs

Other Motivations for Software Test

- Agile processes emphasize test
  - Test driven requirements (user stories)
  - Continual unit testing
- The Industry has created a number of Software Test Certification organizations
  - Some companies highly value these certifications $$
  - Much of the class material this semester will be aligned to the ISTQB foundation level material and test (more later)
- As good as the CMMI is it is particularly weak in the software test function
  - It does address many verification techniques but does not address common methods of software test, estimation, management or planning
- Most CS degree holders graduate without a single class in software test!
  - Only 3 percent of all US colleges/universities offer a class in software testing!

IEEE Computer Society 2022 Report - 23 Computing Technologies That Will Shape the World in 2022

1. **3D Printing**
2. **Big Data and Analytics**
3. Open Intellectual Property Movement
4. Massively Online Open Courses
5. **Security Cross-Cutting Issues**
6. Universal Memory
7. 3D Integrated Circuits
8. Photonics
9. **Cloud Computing**
10. Computational Biology and Bioinformatics
11. Device and Nano-technology
12. Sustainability
13. **High Performance Computing**
14. **The Internet of Things**
15. Life Sciences
16. **Machine Learning and Intelligent Systems**
17. **Natural User Interfaces**
18. **Networking and Inter-connectivity**
19. **Quantum Computing**
20. **Multi-core**
21. **Software Defined Networks**
22. **Robotics**
23. **Computer Vision & Pattern Recognition**

Legend: Software Intensive

Seamless Mesh of Intelligent Devices

IEEE Computer Society 2022 Report - 23 Computing Technologies That Will Shape the World in 2022

Discussion of the software involved technologies

- 3D Printing - ability to change the production line and ability to print biological items (organs)
- Big Data and Analytics - rapid changes in data acquisition, storage, and processing technologies; complex privacy issues.
- Security Cross-Cutting Issues - tradeoff decisions to be made about privacy versus security
- Cloud Computing - Need to develop a “divide-and-conquer” approach to standardizing layers of building block services
- High Performance Computing - enable existing applications to increase parallelism
- The Internet of Things - reach from anywhere to anywhere
- Machine Learning and Intelligent Systems - Intelligent assistants everywhere (e.g. refrigerators) - autonomous cars
- Natural User Interfaces - touch, gesture, and speech need to be integrated to provide a more natural and efficient way of interacting

Recent Worldwide Software Engineering Initiatives

- World-wide initiatives have emerged to promote Software Engineering as a specific body of knowledge with key areas of focus
  - Software intensive technologies continue to grow and directly interact with people, both safety and security concerns arise
  - Software failures have been highly visible and in some cases quite catastrophic https://en.wikipedia.org/wiki/List_of_software_bugs
  - These initiative are intended to address both safety and security concerns and provide more reliable software
- Both the ACM and IEEE have developed specific areas of focus for Software Engineering
  - IEEE Software Engineering Body of Knowledge (SWEBOK) - used to obtain Software Engineering Certifications
  - ACM has developed the Curriculum Guidelines for Graduate Degree Programs in Software Engineering (GSwE2009)

Recent Worldwide Software Engineering Initiatives (cont.)

- Where Are These Initiatives Headed?
  - Originally, the ACM withdrew its support of the IEEE SWEBOK stating that it could not support licensure of software engineers
  - There is considerable controversy over licensure, but many influential leaders are now recanting their criticism due to the growing nature of software intensive systems (e.g., Plante, Cerf)
  - In the US, 40 of 50 states are in the process or have already created a path for the licensure of software engineers - the State of Texas uses the IEEE SWEBOK Certification test
  - Most likely there will not a requirement to obtain a license, but understanding the body of knowledge and having a license will be very beneficial for software engineers to have
  - This really underscores the importance of software engineering curriculum and this class to your career

Software Engineering Already Plays A Key Role In Society And Will Continue To

SWEBOK V3 - 2014

- The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) has been created through cooperation among several professional bodies and members of industry and is published by the IEEE Computer Society (IEEE)
- The standard can be accessed freely from the IEEE Computer Society.
- The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) describes generally accepted knowledge about software engineering.
- Its 15 knowledge areas (KAs) summarize basic concepts and include a reference list pointing to more detailed information.
- For SWEBOK Guide V3, SWEBOK editors received and replied to comments from approximately 150 reviewers in 33 countries.

SWEBOK V3 - 2014 (cont.)

- The published version of SWEBOK V3 has the following 15 knowledge areas (KAs) within the field of software engineering:
  1. Software Requirements
  2. Software Design
  3. Software Construction
  4. Software Testing
  5. Software Maintenance
  6. Software Configuration Management
  7. Software Engineering Management
  8. Software Engineering Process
  9. Software Engineering Models And Methods
  10. Software Quality
  11. Software Engineering Professional Practice
  12. Software Engineering Economics
  13. Computing Foundations
  14. Mathematical Foundations
  15. Engineering Foundations

Curriculum Guidelines for Graduate Degree Programs in Software Engineering

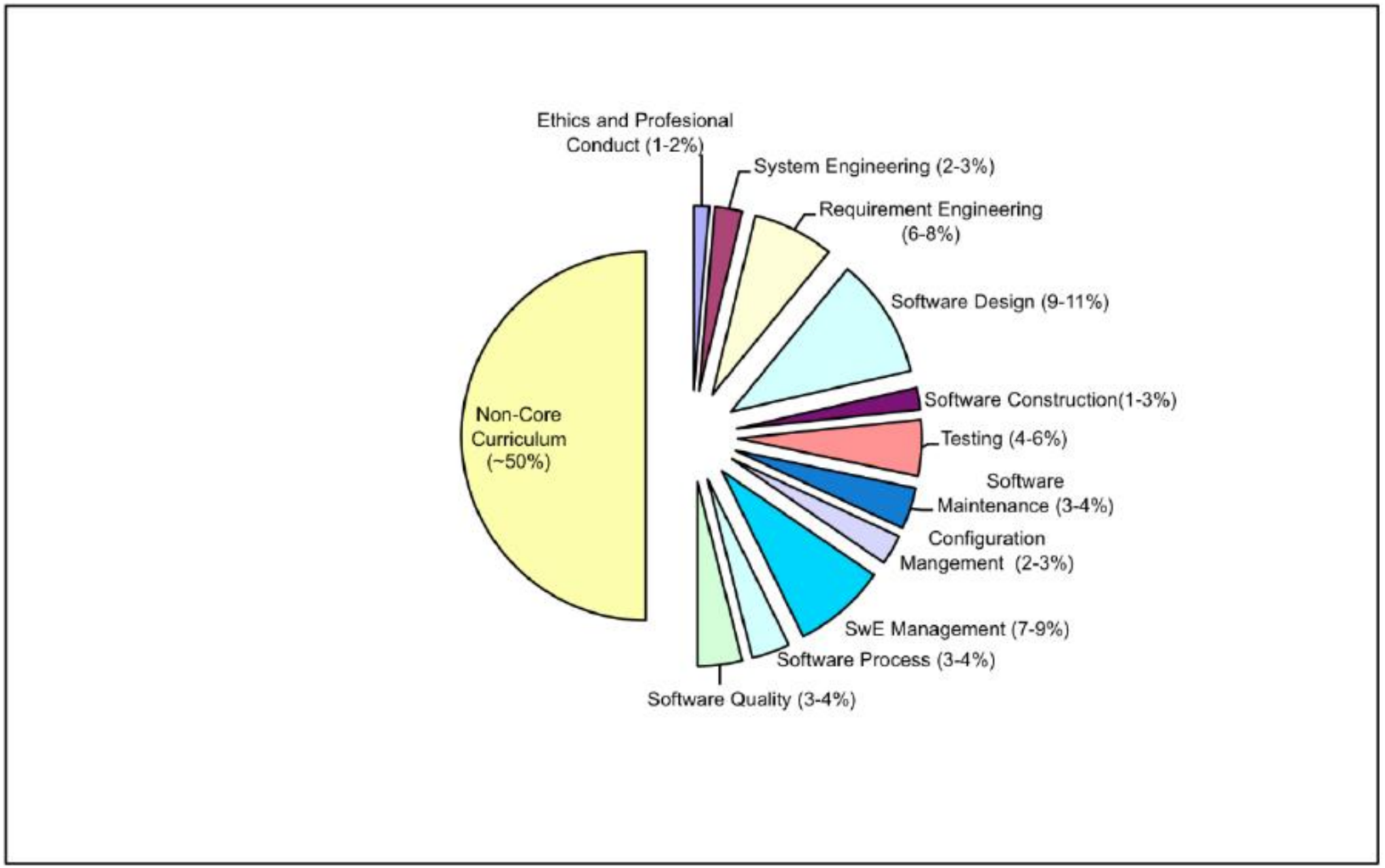
- Curriculum Guidelines for Graduate Degree Programs in Software Engineering - GSwE2009 (published by the ACM) was created to:

- Improve existing graduate programs in SwE from the viewpoint of universities, students, graduates, software builders, and software buyers;

- Enable the formation of new graduate programs in SwE by providing guidelines on curriculum content and advice on how to implement those guidelines; and

- Support increased enrollment in graduate SwE programs by increasing the value of those programs to potential students and employers.

Curriculum Guidelines for Graduate Degree Programs in Software Engineering



Correlation Across Stds

IEEE SWEBOK V3	ACM GSwE2009 CBOK
Software Maintenance	Software Maintenance
Software Engineering Process	Software Engineering Process
Software Testing	Testing
Software Quality	Software Quality
Software Design	Software Design
Software Engineering Management	Software Engineering Management
Configuration Management	Configuration Management
Software Construction	Software Construction
Software Requirements	Requirements Engineering
Software Engineering Professional Practices	Ethics and Professional Conduct
Computing Foundations	Computing Fundamentals
Mathematical Foundations	Mathematical Fundamentals
Engineering Foundations	
Software Engineering Models and Methods	
Software Engineering Economics	
	Systems Engineering
	Software Engineering

A Concrete Example of a Software Bug

Fault: 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;
}
```

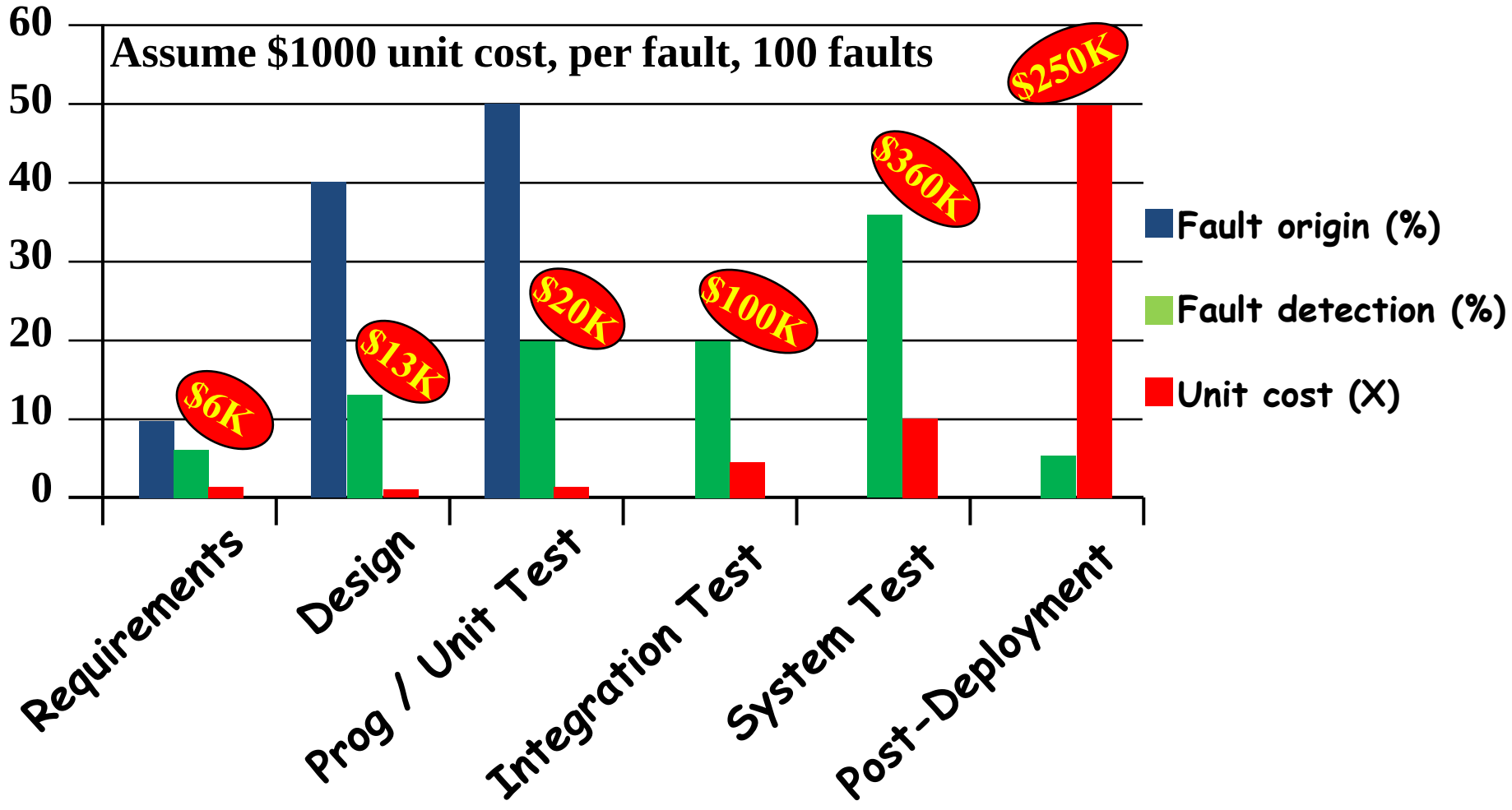
Test 1
[2, 7, 0]
Expected: 1
Actual: 1

Error: i is 1, not 0, on the first iteration
Failure: none

Test 2
[0, 2, 7]
Expected: 1
Actual: 0

Error: i is 1, not 0
Error propagates to the variable count
Failure: count is 0 at the return statement

Cost of Late Testing



Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002

Industry Testing Initiatives

- There is a growing recognition in the industry on the importance of software test
- A number of Software Testing Organizations offer additional training to members
  - The Association for Software Testing – http://www.associationforsoftwaretesting.org/
  - QAI Global Institute - http://www.qaiglobalinstitute.com/
  - International Software Testing Qualification Board (ISTQB) - http://www.istqb.org/
  - American Society for Quality - http://asq.org/index.aspx
  - These address topics such as
    - Foundation level testing
    - Advanced testing concepts
    - Test management principles and practices
- Also a TMMI equivalent to the CMMI - http://www.tmmi.org/

Definition of Terms

- These terms are used throughout the software literature, green are my terms we will use in class
- Bug or Defect or Fault "An incorrect step, process, or data definition in a computer program" - This is in the product
- Error or Mistake: "A human action that produces an incorrect result." - This is what causes a defect
- Failure: "Deviation of a component or system from its expected delivery, service, or result" - This is what the user experiences
- Deficiency: A discovered defect in delivered software
- Latent Defect: An undiscovered defect in delivered software
- Test Oracle: "A source to determine expected results to compare with the actual result of the software under test."

Defect Detection and Removal

- V&V is a defect (fault) detection activity - it is not a failure detection activity.
- The term ‘failure’ is an effect not a cause. One or more defects are the cause of the failure. This means that the same failure may still result when one of its contributing faults is fixed.
- An understanding of defects and removal is important
  - Defects are detected -> some or all detected defects are removed – the removal only occurs when the product is corrected
  - Defect removal is a function of both detection and removal - most software has more defects detected than removed – why?
  - Un-removed defects are called ‘restrictions’
  - Un-detected defects are ‘latent defects’
  - Almost all software has restrictions and latent defects
- We can predict both faults and failures in the software – the latter requires operational history

Verification/Validation and the Software Life Cycle

- The following are definitions from the IEEE SWEBOK V3 section 10.2.2
- Verification : "… ensure[s] that the product is built correctly." Is the product built to its requirements?
- Validation : "… ensure[s] that the right product is built—that is, the product fulfills its specific intended purpose." Are the requirements correct?
- Which of these activities detect defects?
- Common Software Development Life Cycles (SDLCs) include the v-model, prototyping, iterative and incremental development, spiral development, rapid application development, extreme programming and agile.
- In terms of V&V(which is a defect detection activity) most of what we care about from these SDLCs are what activities detect defects and when they occur

Verification/Validation Activities and the SDLC

Note: this does not imply a specific SDLC

Activity	Verification	Validation
Software Requirements	•Requirements Technical Reviews •Requirements Based Testing	Customer Review, Expert Review, Modeling, Prototyping
High Level Design	•High Level Design Technical Reviews •Integration Level Testing	Modeling, Prototyping
Detailed Design	•Detailed Design Technical Reviews •Integration Level Testing	Modeling, Prototyping
Coding	•Code Technical Reviews •Unit Level Testing	Modeling, Prototyping

- **Technical Reviews (SWEBOK) can be Formal Inspections, Walkthroughs, or Peer Reviews**
- **For high maturity teams, the Technical Review activity can detect up to 90 percent of the software defects!**

Software Testing and Techniques Are Essential

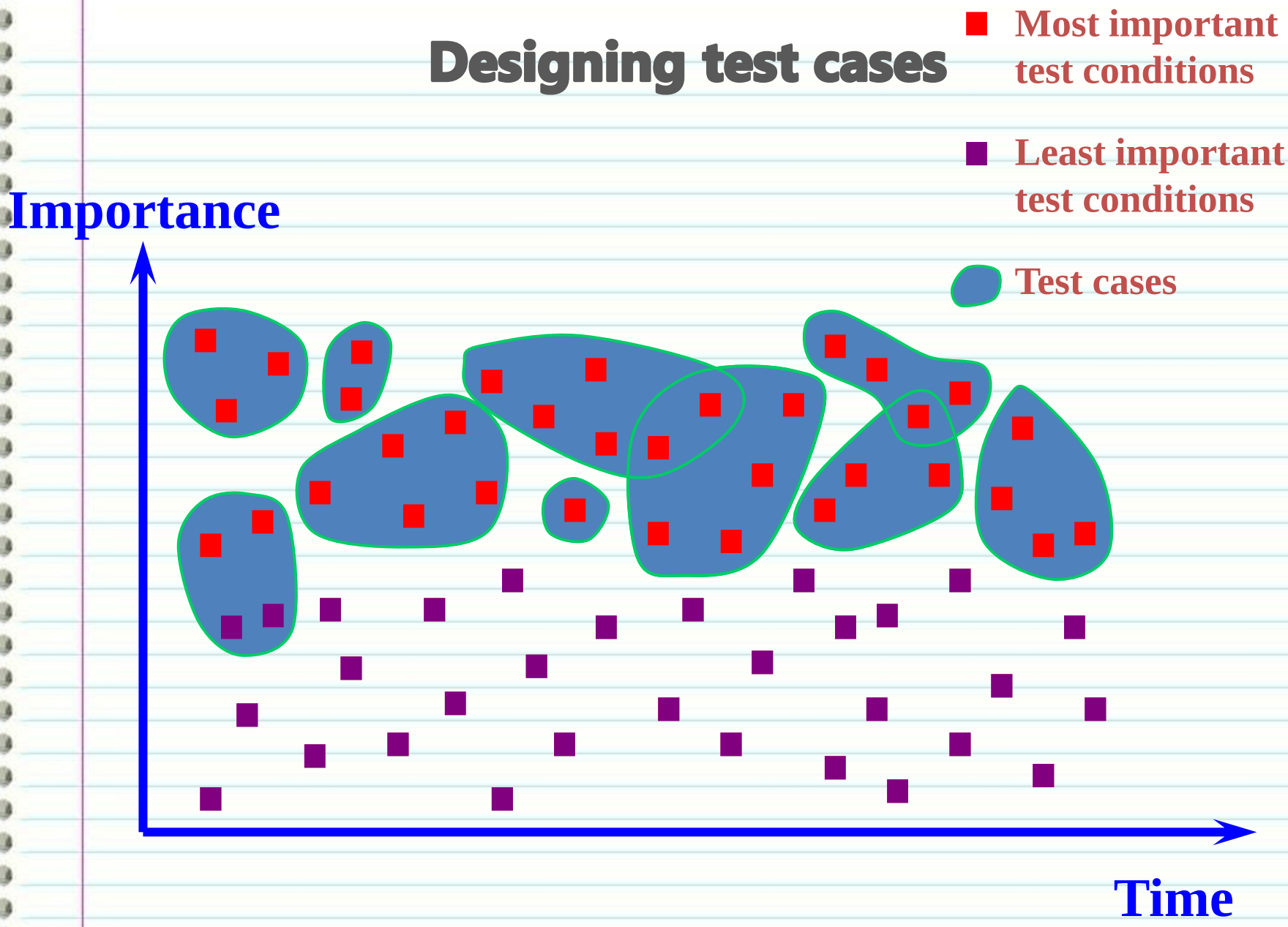
- Technical reviews can detect a significant percentage of defects, but it is important to understand what kinds of defects they detect
  - Defects are typically limited to the scope of the product being reviewed or possibly close neighbors
  - Defects are typically limited to those found statically (as opposed to dynamically such as execution)
- Example: for a "group" of requirements we would normally find the following during a technical review
  - Defects related to the requirements being reviewed (or possibly closely related requirements)
  - Issues with the completeness or correctness of specification (missing/incorrect thresholds, logic, sequencing, etc.)
- Defects that are related to dynamic or more global functionality are best suited for testing – the need for software testing will never be eliminated!
- Software test design
  - Where most defects are detected – not during test execution
  - Techniques are very useful for technical reviews also

Why not just "test everything"?

A simple system –

- **System has 20 screens, with 4 menus per screen and 3 options per menu**
- **Average: 10 fields / screen**
- **2 types input / field**
- **(date as Jan 3 or 3/1)**
- **(number as integer or decimal)**
- **Around 100 possible values**
- **Total for 'exhaustive' testing:**
- **20 x 4 x 3 x 10 x 2 x 100 = 480,000 tests!**

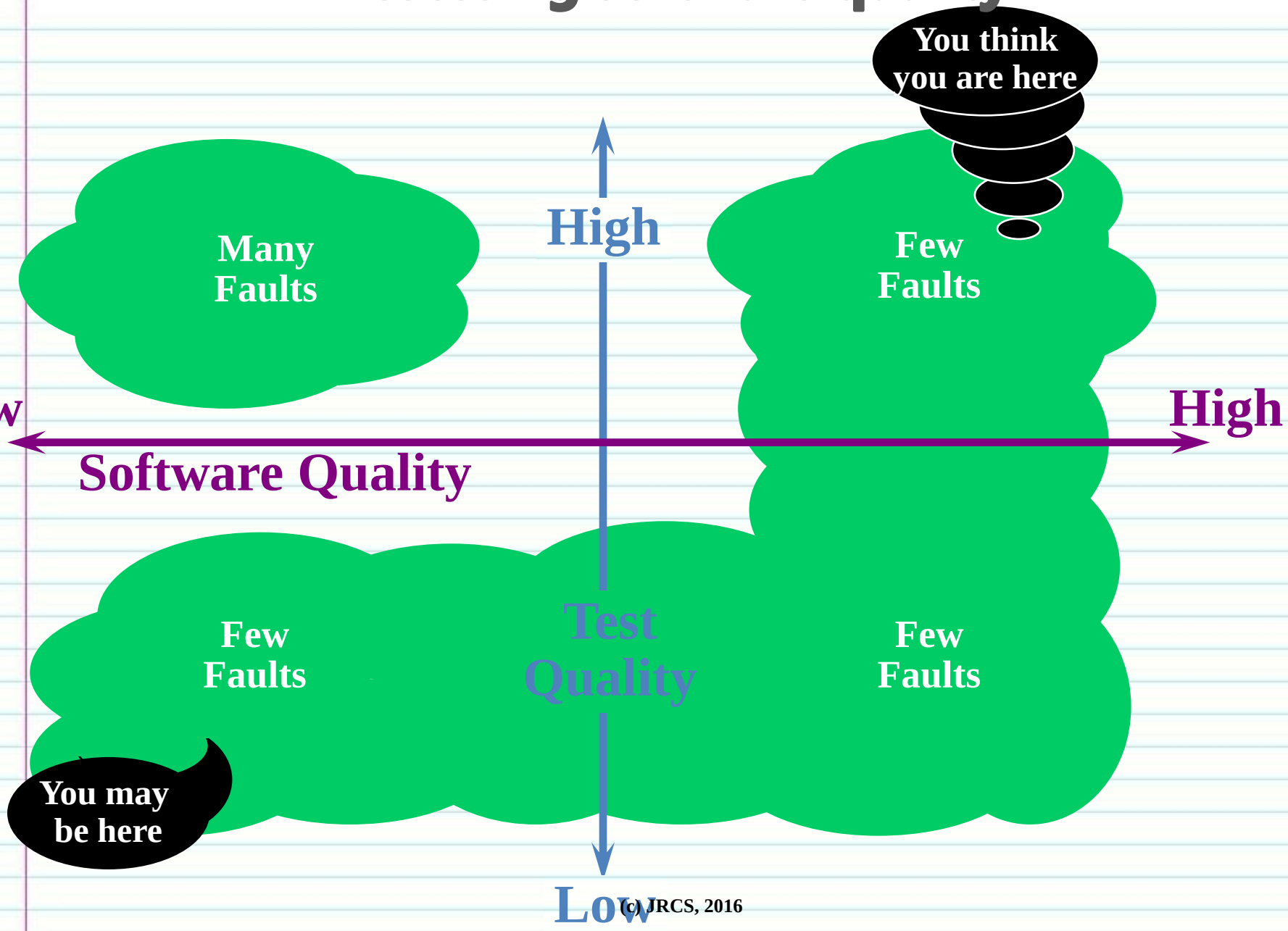
Designing test cases



How to prioritize?

- Possible ranking criteria (all risk based)
  - test where a failure would be most severe
  - test where failures would be most visible
  - test where failures are most likely
  - ask the customer to prioritise the requirements
  - what is most critical to the customer's business
  - areas changed most often
  - areas with most problems in the past
  - most complex areas, or technically critical

Assessing software quality



A Few More Introductory Terms

- Quality: "The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations."
- Homework: How does test affect quality?
- Defect prevention: "A structured problem-solving methodology to identify, analyze and prevent the occurrence of defects." Defect prevention changes the process to prevent occurrence.
- How does this compare with defect detection and removal?
- Homework: What is quality control vs. quality assurance (defect prevention)?
- Debugging: "The process of finding, analyzing and removing the causes of failures in software."
- How does debugging differ from defect detection?

Seven Testing Principles

- 1) Testing shows presence of defects: Testing can show the defects are present, but cannot prove that there are no defects.
- 2) Exhaustive testing is impossible.
- 3) Early testing: In the software development life cycle testing activities should start as early as possible to reduce cost.
- 4) Defect clustering: A small number of modules contains most of the defects discovered. Sometimes referred to as the 80/20 rule.
- 5) Pesticide paradox: If the same kinds of tests are repeated again and again, eventually the same set of test cases will no longer be able to find any new bugs. Does not apply to regression testing.
- 6) Testing is context depending: Different software products have varying requirements, functions and purposes. For example, safety – critical software is tested differently from an e-commerce site.
- 7) Absence-of-errors fallacy: Declaring that a test has unearthed no errors is not the same as declaring the software “error-free”.

A Simple Example - Requirements

- A savings account in a bank has a different rate of interest depending on the balance in the account.

1. 5% rate of interest is given if the balance in the account is between $100 to $1000
2. 3% rate of interest is given if the balance in the account is less than $100
3. 7% rate of interest is given if the balance in the account is at least $1000

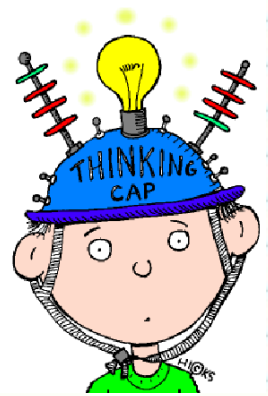
A Simple Example in Java

- Here is some Java code - how many test cases would you use to test this code?

```
2  public class Problem3Class {
3
4      public String getTemp(int temp) {
5          String feels;
6
7          if (temp<=32)
8              feels="Freezing";
9          else {
10             if (temp<40)
11                 feels="Very Cold";
12             else {
13                 if (temp<50)
14                     feels="Cold";
15                 else {
16                     if (temp<80)
17                         feels="Nice";
18                     else
19                         feels="Warm";
20                 }}}
21         return feels;
22     }
23 }
```

Objectives of This Class

- To teach you to think like a tester



- To teach you to break software

- To make you a better software engineer

- To equip you to confidently develop tests day one on the job