Based on Presentations by

- Nurit Gal-oz, Department of Computer Science Ben-Gurion university
- Mira Balaban Department of Computer Science Ben-Gurion university
The Surprising Relationship between Errors Remaining and Errors Found.

![Graph showing the probability of additional errors found as a function of errors already found.](image_url)
A self assessment Test

- Write a set of test cases (i.e., specific sets of data to test a simple program)
- Program description
  - The program reads three integer values from an input dialog
  - The three values represent the lengths of sides of a triangle
  - The program displays a message that states whether the triangle is

- equilateral triangle
- isosceles triangle
- scalene triangle
Scalene, Isosceles, Equilateral

- A scalene triangle is one where no two sides are equal
- Isosceles has two equal sides
- Equilateral has three sides equal length
- Angles opposite the equal sides in an isosceles also are equal
Evaluate test cases: 1

- Give yourself one point for each “yes” answer
  1. Do you have a test case that represent a valid scalene triangle? (note test cases such as 1,2,3, and 2,5,10 do not warrant a “yes”)
  2. Do you have a test case that represents a valid equilateral triangle?
  3. Do you have a test-case that represents a valid isosceles triangle (2,2,4 would not count because it is not a valid triangle)
  4. Do you have at least three test cases that represent valid isosceles triangles such that you have tried all three permutations of two equal sides (such as 3,3,4; 3,4,3, 4,3,3)?
  5. Do you have a test case in which one side has a zero value?
  6. Do you have a test case in which one side has a negative value?
  7. Do you have a test case with three integers greater than zero such that the sum of two of the number is equal to the third? (e.g. 1,2,3)
Evaluate test cases: 2

8. Do you have at least three test cases in category 7 such that you have tried all three permutations where the length of one side is equal to the sum of the lengths of the other two sides (e.g. 1,2,3; 1,3,2; and 3,1,2)?

9. Do you have a test case with three integers greater than zero such that the sum of two of the numbers is less than the third (e.g. 1,2, 4 or 12, 15, 30)?

10. Do you have at least three test cases in category 9 such that you have tried all three permutations (e.g. 1,2,4: 1,4,2; and 4,1,2)?

11. Do you have a test case in which all sides are zero (0,0,0)?

12. Do you have at least one test case specifying non-integer value (such as 2.5, 3.5, 5.5)?

13. Do you have at least one test case specifying the wrong number of values (two rather than three integers)

14. For each test case did you specify the expected output from the program in addition to the input values
Evaluation results

- Highly qualified professional programmers score (on average) 7.8 out of possible 14
- Congratulations if you have done > 7.8
- Now consider testing a 100,000 statement air traffic control system, a compiler, or even payroll system, C++ programs
  - Bottom line: complete testing of real-world program is very difficult

  test
Testing...
Software verification and validation

- **Verification and validation** is intended to show that a system conforms to its **specification** and meets the **requirements** of the system customer.
- Involves **checking** and **review processes** and **system testing**.
- **System testing** involves executing the system with test cases that are derived from the **specification of the real data** to be processed by the system.
Verification vs. Validation

- Verification:
  "Are we building the product right?"
  - The software should conform to its specification.

- Validation:
  "Are we building the right product?"
  - The software should do what the user really requires.
Verification & Validation Goals

Why Test?

- Establish confidence that the software is fit for purpose.
- Demonstrate conformance to requirements
- Find faults
- Reduce costs
- Show system meets user needs
- Assess the software quality
Verification & Validation Goals

- Establish **confidence** that the software is fit for purpose.
- Two principal objectives
  - The **discovery of defects** in a system
  - The assessment of whether or not the system is **useful and useable** in an operational situation.
- This **does not mean:**
  - completely free of defects
  - correct
- Rather, it must be **good enough for its intended use** and the type of use will determine the degree of confidence that is needed.
V & V confidence

- Depends on system’s purpose, user expectations and marketing environment
  - **Software function**
    - The level of confidence depends on how **critical** the software is to an organization.
  - **User expectations**
    - Users may have low expectations of certain kinds of software.
  - **Marketing environment**
    - Getting a product to market early may be more important than finding defects in the program.
Static and dynamic verification

- **Static verification.** Concerned with Analysis of the static system representation to **discover problems**
  - **Software inspections**
    - May be supplement by tool-based document and code analysis
  - **Static analysis**
  - **Formal verification**

- **Dynamic Validation:** Concerned with exercising and observing product behavior, using **Software testing**.
  - The system is executed with **test data** and its operational behavior is observed
Static verification
Code Inspections

- Formalized approach to document reviews
- Intended explicitly for defect detection (not correction).
- Defects may be: logical errors, anomalies in the code that might indicate an erroneous condition (e.g. an uninitialized variable) or non-compliance with standards.
- Inspections can check conformance with a specification but not conformance with the customer’s real requirements.
- Non-functional characteristics:
Static verification

Code Inspections

• **Non-functional characteristics**: Inspections **cannot** check non-functional characteristics such as performance, usability, etc.

• Usually done by a group of programmers

The comments are to the program not the programmer!!
Static verification

Code Inspections

- An Error Checklist for Inspections
  - The checklist is largely language independent
  - Sometimes (unfortunately) concentrate more on issues of style than on errors
    - for example, “Are comments accurate and meaningful?” and “Are if-else, code blocks, and do-while groups aligned?”, and the error checks are too nebulous to be useful (such as “Does the code meet the design requirements?”).

- Beneficial side effects
  - The programmer usually receives feedback concerning programming style, choice of algorithms, and programming techniques.
  - The other participants gain in a similar way by being exposed to another programmer’s errors and programming style.
## Static verification

### Code Inspections

### Table 3.1: Inspection Error Checklist Summary, Part I

<table>
<thead>
<tr>
<th>Data Reference</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Unset variable used?</td>
<td>1. Computations on nonarithemetic variables?</td>
</tr>
<tr>
<td>3. Non integer subscripts?</td>
<td>3. Computations on variables of different lengths?</td>
</tr>
<tr>
<td>4. Dangling references?</td>
<td>4. Target size less than size of assigned value?</td>
</tr>
<tr>
<td>5. Correct attributes when aliasing?</td>
<td>5. Intermediate result overflow or underflow?</td>
</tr>
<tr>
<td>6. Record and structure attributes match?</td>
<td>6. Division by zero?</td>
</tr>
<tr>
<td>Passing bit-string arguments?</td>
<td></td>
</tr>
</tbody>
</table>

### Table 3.2: Inspection Error Checklist Summary, Part II

<table>
<thead>
<tr>
<th>Control Flow</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Multway branches exceeded?</td>
<td>1. File attributes correct?</td>
</tr>
<tr>
<td>2. Will each loop terminate?</td>
<td>2. OPEN statements correct?</td>
</tr>
<tr>
<td>4. Any loop bypasses because of entry conditions?</td>
<td>4. Buffer size matches record size?</td>
</tr>
<tr>
<td>5. Are possible loop fall-throughs correct?</td>
<td>5. Files opened before use?</td>
</tr>
<tr>
<td>6. Off-by-one iteration errors?</td>
<td>6. Files closed after use?</td>
</tr>
<tr>
<td>7. DO-END statements match?</td>
<td>7. End-of-file conditions handled?</td>
</tr>
<tr>
<td>8. Any nonexhaustive decisions?</td>
<td>8. I/O errors handled?</td>
</tr>
</tbody>
</table>

### Interfaces

| Number of input parameters equal to number of arguments? | 1. Any unreferenced variables in cross-reference listing? |
| Parameter and argument attributes match? | 2. Attribute list what was expected? |
| Parameter and argument units system match? | 3. Any warning or informational messages? |
| Number of arguments transmitted to called modules equal to number of parameters? | 4. Input checked for validity? |
| Attributes of arguments transmitted to called modules equal to attributes of parameters? | 5. Missing function? |
| Units system of arguments transmitted to called modules equal to units system of parameters? | 6. Missing function? |
| Number, attributes, and order of arguments to built-in functions correct? | 7. Number, attributes, and order of arguments to built-in functions correct? |
| Any references to parameters not associated with current point of entry? | 8. Any references to parameters not associated with current point of entry? |
| Input-only arguments altered? | 9. Input-only arguments altered? |
| Global variable definitions consistent across modules? | 10. Global variable definitions consistent across modules? |
| Constants passed as arguments? | |

---

Software Engineering, Software Testing 18
Static verification
Automated Static Analysis

- Static analyzers are software tools for source text processing.
- A static Analyzer:
  - A collection of algorithms and techniques used to analyze source code in order to automatically find bugs
  - Parses the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team.
- Very effective as an aid to inspections – they are a supplement to but not a replacement for inspections.
Static verification
Automated Static Analysis

```java
Method invocation 'fromCurrency.equals("USD")' may produce 'java.lang.NullPointerException'.

double answer = 0;
if (fromCurrency.equals("USD") && toCurrency.equals("CDN")) {
    Assert 'fromCurrency != null'
}

attribute = parseAttribute(isempty, asp, php);
if (attribute == null) {
    return;
}
value = parseValue(attribute, false, isempty, delim);
if (attribute != null) {
    // Condition 'attribute != null' is always 'true'.
}
else {
    av = new AttrVal(null, null, null, null,
            0, attribute, value);
    Report.attrError(this, this.token, value,
            Report.BAD_ATTRIBUTE_VALUE);
}
```
Static verification

Formal Verification Methods

- Formal methods can be used when a mathematical specification of the system is available.
- Form the ultimate static verification technique.
- Involve detailed mathematical analysis of the specification.
- May develop formal arguments that a program conforms to its mathematical specification.

Diagram:
- Model (System Design) → Model Checking Tool → Formal Specification
- Output - Does the model satisfied Specification
Static verification
Arguments for/against formal methods

For

• Producing a mathematical specification requires a detailed analysis of the requirements and this is likely to uncover errors.
• Can detect implementation errors before testing when the program is analyzed alongside the specification

Against

• Require specialized notations that cannot be understood by domain experts.
• Very expensive to develop a specification and even more expensive to show that a program meets that specification.
• It may be possible to reach the same level of confidence in a program more cheaply using other V & V techniques.
Software testing

- Yet, in spite of this checkout expense, delivered “verified” and “validated” code is still **unreliable**.

Software Testing

- The **only** validation technique for *non-functional requirements* as the software has to be executed to see how it behaves.
- Should be used in conjunction with static verification to provide full V&V coverage.
Good Software Tester

- Technical Skills
- Analytical Skills
- Passion
- Attitude
- Verbal and Written Communication
- Productivity

Good Software Tester
Testing

When?
- Testing along software development process.

What?
- What to test?

How?
- How to conduct the test?
- Discussion on Integration testing
The Testing Process

- Unit testing
- Module testing
- Sub-system testing
- System testing
- Acceptance testing

Component testing
Integration testing
User testing
Test Planning

- Test planning is concerned with **scheduling and resourcing** all of the activities in the testing process.
- It involves defining the **testing process**, taking into account the people and the time available.
- Usually, a test plan will be created, which defines what is to be tested, the predicted testing schedule, and how tests will be recorded.
- For critical systems, the test plan may also include details of the tests to be run on the software.
Software Testing Features

- The scope of testing
  - The different levels of the system that testing addresses
- Test techniques
  - Some of the approaches to building and applying tests
- Test management
  - How we manage the testing process to maximize the effectiveness and efficiency of the process for a given product
Testing scope

- **Testing in the small – unit testing**
  Exercising the smallest executable units of the system

- **Testing the build – integration testing**
  Finding problems in the interaction between components

- **Testing in the large – system testing**
  Putting the entire system to test
Black-box and White-box Testing

**BLACK BOX TESTING APPROACH**

**WHITE BOX TESTING APPROACH**
White-box Testing

- White-box Testing
  - Testing the degree to which test cases cover the logic
    - Also named: open-box, clear box, glass box, logic-driven
  - Examine the internal structure of the program.
  - Test all possible paths of control flow (theoretically)
  - Capture:
    - errors of omission – neglected specification
    - errors of commission – not defined by the specification
Black-box Testing

- Black-box testing
  - Also named: closed box, data-driven, input/output driven, behavioral testing
  - Provide both valid and invalid inputs.
  - The output is matched with expected result (from specifications).
  - The tester has no knowledge of internal structure of the program.
  - It is impossible to find all errors using this approach.
  - Test all possible types of inputs (including invalid ones like characters, float, negative integers, etc.).
Testing Stages

- **Unit testing**
  - Individual components are tested.

- **Module testing**
  - Related collections of dependent components are tested.

- **Sub-system testing**
  - Modules are integrated into sub-systems and tested. The focus here should be on interface testing.

- **Black Box**
  - **Acceptance testing**
    - Testing with customer data to check that it is acceptable.
  - **System testing**
    - Testing of the system as a whole.

Component Testing

Integration Testing

User Testing

System Testing
System/Release testing (1)

- Testing a release of a system that will be distributed to customers.
- **Primary goal:**
  
  *Increase the supplier’s confidence that the system meets its requirements.*

- Release testing is usually *black-box or functional* testing
  - Based on the system specification only;
  - Testers do not have knowledge of the system implementation

- Part of release testing may involve testing the emergent properties of a system, such as *performance* and *reliability* (System testing).
System Testing (1)

- **Usability Testing**
  - Has each user interface been tailored to the intelligence, educational background, and environmental pressures of the end user?
  - Are the outputs of the program meaningful and devoid of computer gibberish?

**Usability testing**

Can users use my app?
Usability Testing
System Testing (1)

- **Usability Testing**
  - Has each user interface been tailored to the intelligence, educational background, and environmental pressures of the end user?
  - Are the outputs of the program meaningful, nonabusive, and devoid of computer gibberish?

- **Security Testing**
  - Attempting to devise test cases that subvert the program’s security checks.
  - For example, get around an operating system’s memory protection mechanism.
  - You can try to subvert a database management system’s data security mechanisms.

- **Volume Testing**
  - Heavy volumes of data
  - The purpose of volume testing is to show that the program cannot handle the volume of data specified in its objectives
  - For instance, a compiler could be fed an absurdly large source program to compile.
System Testing (2)

- **Stress Testing**
  - Subjects the program to heavy loads, or stresses.
  - The tester attempts to **stress or load an aspect of the system to the point of failure**
  - The tester identifies **peak load conditions** at which the program will fail to handle required processing loads within required time spans.
  - Stress testing has **different meaning for different industries** where it is used.
  - Stressing the system often causes defects to come to light.
  - Stressing the system test **failure behavior**.
  - Systems should not fail catastrophically.
  - It is applicable, however, to programs that operate under varying loads, or interactive, real-time, and process control programs.
System Testing (3)

- **Performance Testing**
  - Determine if the system meets the **stated performance criteria**: response times and throughput rates under certain workload and configuration conditions.
  - E.g. A Login request shall be responded to in 1 second or less under a typical daily load of 1000 requests per minute.

- **Recovery Testing**
  - Programs such as operating systems, database management systems, have recovery objectives that state how the system is to recover from programming errors, hardware failures, and data errors.
  - **Programming errors can be purposely injected into a system** to determine whether it can recover from them.
  - Hardware failures such as memory parity errors or I/O device errors can be simulated.
  - Data errors such as noise on a communications line or an invalid pointer in a database can be created purposely or simulated to analyze the system’s reaction.
Regression Testing

- When you are doing an enhancement:
  - you should run all old tests
  - even ones that are invalid for original purpose;
  - just provide new expected results.

- An expensive process. Requires **automation**.
  - Can apply risk analysis to reduce the amounts of tests.

Regression: "when you fix one bug, you introduce several newer bugs."
Regression Testing

- Adding new or changing module impacts the system
  - New data flow paths established
  - New I/O may occur
  - New control logic invoked
- Regression testing is re-execution of subset of tests that have already been conducted
  - Ensures changes have not propagated unintended side effects