Topic 2: The Security Analysis Process

1. Security analysis as a formal process
2. Security policies
3. Risk analysis, measurement and management
4. Software security measurement
5. Software testing

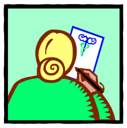
Notes supplied by David Smith (GeorgiaTech), Harry Erwin (University of Sunderland) and Software Engineering Institute, Carnegie-Mellon University are used

Security Analysis is a Formal Process.

1. Start by identifying the system and items to be protected.
2. Identify the security policies that must be enforced.
3. Define the trust relationships to be supported.
4. Perform a risk analysis to identify the threats.
5. Establish the assumptions of secure operation.
6. List the security objectives you must implement.
7. Define the resulting security functions to be provided.
8. Provide the detail, implement, and test.

What are the Items to be Protected?

- If you try to protect everything, you protect nothing.
- Can include data, equipment, reputation, and resources.
- Take into account time. Any security can be broken with enough time and resources. Provide alarms and appropriate responses.



Security Policies

- This is where one should start
- Be realistic.
- If you can’t secure what you *need* to secure, perhaps you should reconsider building the system.

Trust Relationships

- You must understand trust to be able to secure a system and have the system be usable.
- Trust should influence your security architecture.

Risk Analysis

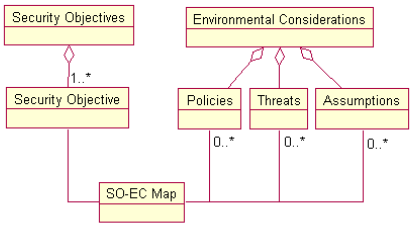
Be realistic.

- Hackers are creative—don’t reject vulnerabilities out of hand. “Security by Obscurity” is not a solution.
- Don’t try to protect everything, but don’t leave obvious holes.

Assumptions of Secure Operation

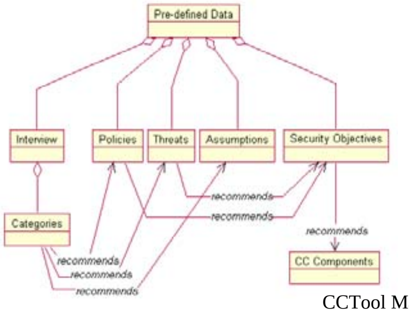
- Should be associated with *individual* security objectives to help you select your security requirements.

The Security Mapping Process



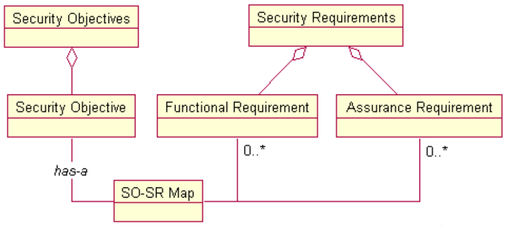
CCTool Manual

Security Analysis Relationships



CCTool Manual

Security Objectives Result in Security Requirements



CCTool Manual

Security Objectives

- “The results of the analysis of the security environment can then be used to state the security objectives that counter the identified threats and address identified organizational security policies and assumptions. The security objectives should be consistent with the stated operational aim or product purpose of the system, and any knowledge about its physical environment.”

Intent of the Objectives

- “The intent of determining security objectives is to address all of the security concerns and to declare which security aspects are either addressed directly by the system or by its environment. This categorization is based on a process incorporating engineering judgment, security policy, economic factors and risk acceptance decisions.”

Sources of Security Functional Requirements

- "The CC and the associated security functional requirements are not meant to be a definitive answer to all the problems of IT security. Rather, the CC offers a set of well understood security functional requirements that can be used to create trusted products or systems reflecting the needs of the market. These security functional requirements are presented as the current state of the art in requirements specification and evaluation."

How do You Use the Security Requirements?

- The functional requirements describe *what* your security solution must do to allow you to operate safely. These can be compared to the functionality provided by vendor software and hardware.
- The assurance requirements describe what development practices and testing the vendor must follow to assure the users that the functionality provided *actually works*.

Role of Security Testing

- "Security requirements generally include both requirements for the presence of desired behavior and requirements for the absence of undesired behavior. It is normally possible to demonstrate, by use or testing, the presence of the desired behavior. It is not always possible to perform a conclusive demonstration of absence of undesired behavior. Testing, design review, and implementation review contribute significantly to reducing the risk that such undesired behavior is present."

Risk Measurement and Management

- Assessment
  - Identification
  - Analysis
  - Prioritization
- Control
  - Planning
  - Resolution
  - Monitoring

What is a risk and why do I care?

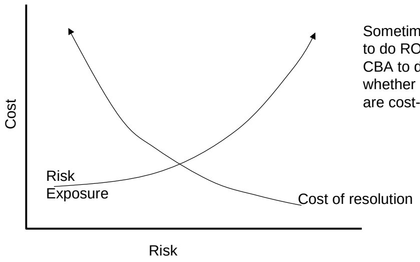
I am used to thinking 3 or 4 months in advance about what I must do, and calculate on the worst. If I take so many precautions it is because it is my custom to leave nothing to chance. -- Napolean I 1808

- A potential problem
- Risk has 2 parts:
  - Probability (Likelihood)
  - Consequence (Loss)
- Risk results in Exposure : Frequently referred to as L^2

Types of Risk

- Project
  - Operational
  - Organizational
  - Contractual
- Process
  - Management
  - Technical
- Product

We can't do everything



Sometimes we need to do ROI or CBA to determine whether risk measures are cost-effective

Characteristics of Good Risk Mgt

- Proactive
- Integrated
- Systematic (20/80 rule – ID::Control)
- Disciplined (P^2I^2)
  - People
  - Process
  - Infrastructure
  - Implementation

Levels of Risk Management

- Crisis Management (Fire Fighting)
- Fix on Failure
- Risk Mitigation
- Risk Prevention
- Elimination of root causes

Risk Identification

- Conduct a risk assessment. (Formal, interviews, facilitated meetings)
- Identify risk systematically. (Checklists)
- Define risk attributes (L^2) (Qualitative).
- Document identified risk.
- Communicate identified risk.

Risk Analysis

- Group similar and related risks
- Determine risk drivers
- Determine source of risk (root cause)
- Use risk analysis techniques and tools
- Estimate risk exposure (Quantitative)
- Evaluate risk against criteria (Severity, Time)
- Rank risks relative to other risks (Top-n)

Risk Analysis Techniques

- Causal Analysis (Cause-Effect)
- Decision Analysis
  - Decision Tree
  - Influence Diagram
- Gap Analysis
  - Magnitude
  - Radar Charts
- Pareto Analysis
- Sensitivity Analysis
- Technical Models, Prototypes
- COCOMO II http://sunset.usc.edu/research/COCOMOII/index.html

Top-n List Format

RISK	Current Priority	Previous Priority	Weeks on Top 10	Action Plan Status	Risk Rating
High SW productivity rate	1	2	2	Capturing requirements into requirements DB tool. Ensuring availability of adequate personnel resources	High
Off-site SW development	2	9	2	Increasing travel budget for additional site reviews. Setting up network access capability.	High

Risk Planning

- Develop scenarios for high-severity risks
- Develop resolution alternatives
- Select resolution approach
- Develop risk action plan
- Establish thresholds for early warning
  - When should I start getting worried?

Risk Scenario

- Think about risk as if it has occurred
- State the sequence of events
- List the events and conditions that would precede risk occurrence.

Risk Resolution Alternatives

- Acceptance
- Avoidance (Eliminate)
- Protection (Redundancy)
- Reduction (Mitigation, Prevention, Anticipation)
- Research (Need more info)
- Reserves (Slush fund, bank, pad)
- Transfer (shift to someone else)

Do Risk resolution for home PC against lightning.

Selection Criteria

- Picking a cost effective strategy
  - Risk Leverage (cost-benefit) =

    $RE_{(before)} – RE_{(after)}$ / Resolution Cost
  - ROI = ΣSavings/Cost
- Diversification
  - Don’t put all the eggs in one basket.

Risk Tracking

- Monitor risk scenarios
- Compare thresholds to status
- Provide notification for triggers
- Report risk measures and metrics

Risk Resolution

- Respond to notification of triggering event
- Execute risk action plan
- Report progress against plan
- Correct for deviations from plan

Risk is Risky Business

Risk is inherent in the development of any large software system. A common approach to risk is to simply ignore it.

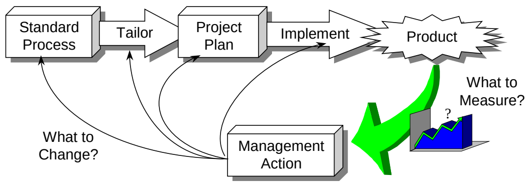
Those who choose to minimize or avoid risk, as opposed to manage it, are setting a course for obsolescence.

If you don't actively attack risks, they will actively attack you.

Whatever can go wrong will go wrong, and at the worst possible time.

If you have brainstormed 4 ways that your project can fail, a new 5th way will present itself.

The Software Security Management Challenge



Measurements

- Product Metrics
  - Work Products
- Management Metrics
  - Work completion
- Process Metrics
  - Work Quality

Product Metrics

Determine the progress of the software development

- Requirements Volatility
- Requirements Quality
- Comparing Actuals to Estimates
  - Requirements count
  - Resource Usage
  - Code size (Function Points)
  - Tests complete

Management Metrics

- Comparing Actuals to the Plan
  - Schedule Performance Index (SPI):

$$\frac{\text{Budgeted Cost of Work Performed (BCWP)}}{\text{Budgeted Cost of Work Scheduled (BCWS)}}$$

  - Cost Performance Index (CPI):

$$\frac{\text{Actual Cost of Work Performed (ACWP)}}{\text{Budgeted Cost of Work Scheduled (BCWS)}}$$

- Value of Indices
  - Management visibility into overall performance
- Hazards of Indices
  - Not a good indicator of emerging problems
  - Large, long-term program can be in significant trouble before indicators show deviation (e.g. batting average)

Metrics Overview

I. Development Progress
- 1 Requirements Allocated
- 2 Components Designed
- 3 Components Implemented
- 4 Components Integrated
- 5 Requirements Tested
- 6 Paths Tested
- 7 Test Cases Completed

II. Stability
- 1 Requirements Volatility
- 2 Problem Report Trends
- 3 Problem Report Aging
- 4 Resource Availability
- 5 Defect Profile
- 6 Rework Effort

III. Personnel
- ***1 Staff Level***
- ***2 Staff Turnover***

IV. Product Size
- ***1 Number of Components***
- ***2 Function Points***
- ***3 Lines of Code***
- ***4 Words of Memory***

V. Resource Consumption
- ***1 CPU Utilization***
- ***2 CPU Throughput***
- ***3 I/O Utilization***
- ***4 I/O Throughput***
- ***5 Memory Utilization***
- ***6 Storage Utilization***

VI. Cost & Schedule
- ***1 Progress***
- ***2 SPI***
- ***3 CPI***

MEASUREMENT SPECIFICATION

- Data Items - for each measure, identify all
  - data elements, and
  - levels of collection and reporting
- Data Types - plans, changes to plans, and actuals for each measure should be collected, reported, and updated regularly.
- Measurement Definitions - for each measure, identify
  - definitions and methodologies that will be used
  - differences between the estimation methodologies and the way the actuals are counted, and
  - "exit" criteria for counting actuals.
- Data Dates - identify both the date that the data were collected and the date that they are reported.

MEASUREMENT SPECIFICATION (cont)

- Collection Timing - on a periodic, not event driven basis.
- Measurement Scope - If more than one organization is involved in SW development,
  - collect from each and identify the source
  - unify definitions for the same measures
- Program Phase - The measures should generally be applied to all life cycle phases, including program planning, development, and software support.
- Reporting Mechanisms - The mechanisms for reporting data to management and the customer

Testing

- Background
- Dynamic and Static Testing
- Testing from the Bottom Up
- Test Plans
- White and Black Box Testing
- Special Testing - Real-Time Systems

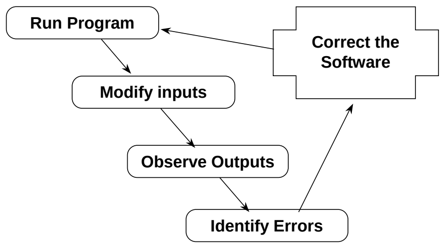
Testing Principles

The goal of testing is not to prove that software is error-free, but rather just to find what errors we can

- All tests should be traceable to customer requirements
- Tests should be planned long before testing begins
- The Pareto Principle applies to software testing
- Testing should begin "in the small" and progress towards testing "in the large"
- Exhaustive testing is not possible
- Testing should be conducted by a third party

Verification:"Are we building the product right?"
Validation: "Are we building the right product?"

Dynamic Testing

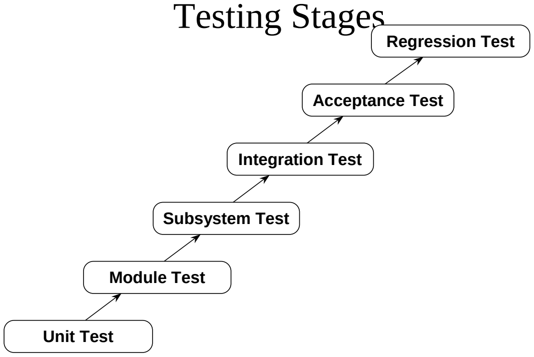


Static Testing

- Program Inspections
- Analysis
- Formal Verification

- **Can reduce cost**
- **Unlikely to discover difficult errors:**
  - ***requirements errors***
  - ***errors of omission***
- **Only useful on small systems or unit testing**

Testing Stages



Unit Test

Module Test

Subsystem Test

Integration Test

Acceptance Test

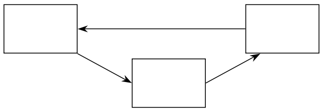
Regression Test

Unit Test

The FIRST and MOST EXHAUSTIVE Test

- Nothing else will work right unless this is done well
- Individual components tested in isolation
  - Procedure,
  - function,
  - object
- Stand-alone entities
- Check that component meets spec

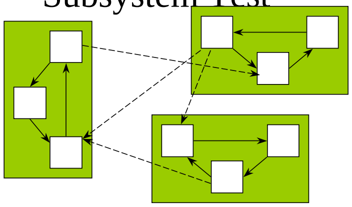
Module Test



- Modules identified during system design
- Combine interdependent components

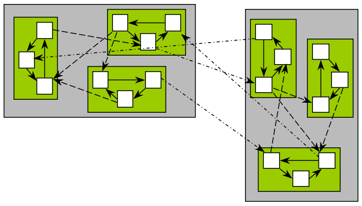
  (initially, tested units)
- Test interaction of related components
- Modules are stand-alone, entities

Subsystem Test

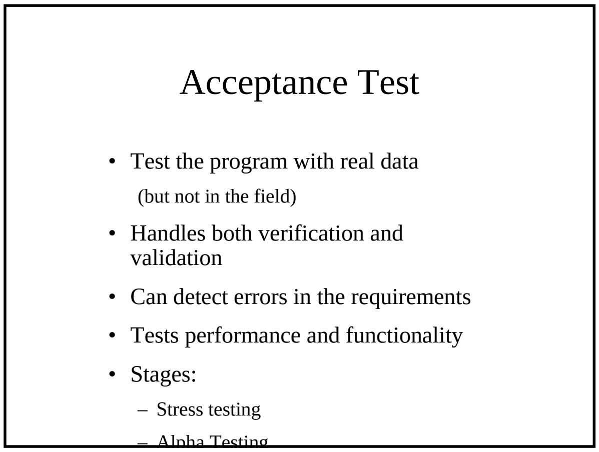


- Combine related modules
- Rigorously exercise interfaces
- Detect interface mismatches

Integration Test

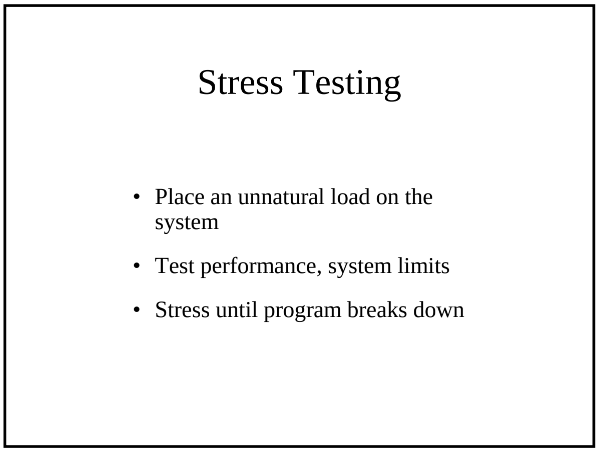


- Combine (potentially unrelated) subsystems
- Find unanticipated interactions between components of subsystems
- Validate the overall functionality of the system



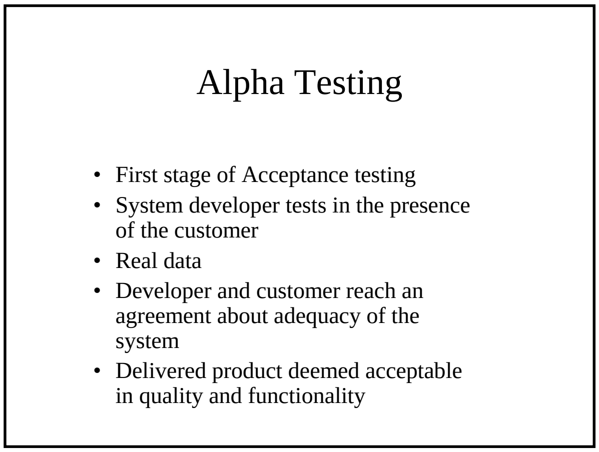
Acceptance Test

- Test the program with real data
  (but not in the field)
- Handles both verification and validation
- Can detect errors in the requirements
- Tests performance and functionality
- Stages:
  - Stress testing
  - Alpha Testing



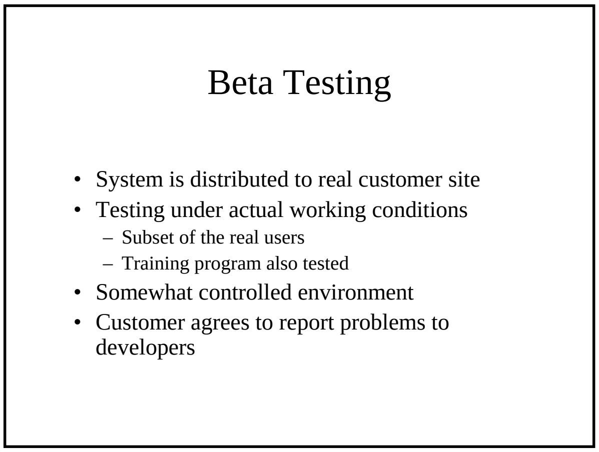
Stress Testing

- Place an unnatural load on the system
- Test performance, system limits
- Stress until program breaks down



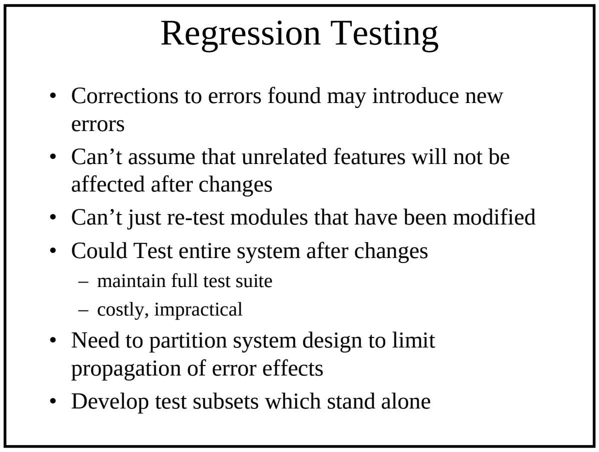
Alpha Testing

- First stage of Acceptance testing
- System developer tests in the presence of the customer
- Real data
- Developer and customer reach an agreement about adequacy of the system
- Delivered product deemed acceptable in quality and functionality



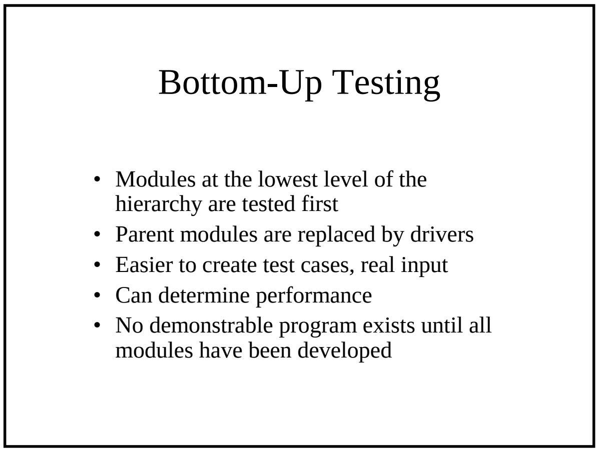
Beta Testing

- System is distributed to real customer site
- Testing under actual working conditions
  - Subset of the real users
  - Training program also tested
- Somewhat controlled environment
- Customer agrees to report problems to developers



Regression Testing

- Corrections to errors found may introduce new errors
- Can’t assume that unrelated features will not be affected after changes
- Can’t just re-test modules that have been modified
- Could Test entire system after changes
  - maintain full test suite
  - costly, impractical
- Need to partition system design to limit propagation of error effects
- Develop test subsets which stand alone



Bottom-Up Testing

- Modules at the lowest level of the hierarchy are tested first
- Parent modules are replaced by drivers
- Easier to create test cases, real input
- Can determine performance
- No demonstrable program exists until all modules have been developed

Top-Down Testing (Prototyping)

- Start at subsystem level - replace modules with stubs
- Modules can be tested as soon as they are coded
- Top-down detects design errors early
- A working system exists at all times
- Test output is artificial

Test Plans

- Testing can consume half of the overall development costs
- Test plans describe the testing process
- Components of a test plan:
  - Major phases of testing
  - Traceability to requirements
  - Schedule and resource allocation
  - Relationship between test plan and other documents
  - Test auditing

Test Cases

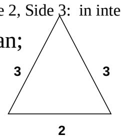
- Not the same thing as test data
- Test Cases:
  - input and output specifications
  - statement of the function under test
  - mapping to requirements

Test Cases

<u>Example</u>: A program that determines whether a triangle is isosceles:

- function Is Isosceles

  (Side 1, Side 2, Side 3: in integer)
- return boolean;



Test Cases

How many test cases are there?

- A triangle that is isosoles (2,2,3)
- Reorder the equivalent sides (2,3,2) (3,2,2)
- Triangle that is equilateral (2,2,2)
- Triangle that is not isosoles (1,2,3,)
- Reorder numbers (2,3,1)
- Boundary conditions (1,2,0)
- Reorder boundaries (1,0,2) (0,1,2)
- Multiple boundaries (0,0,1)
- All boundaries (0,0,0)
- Large numbers (6500001, 4, 35467843)

Black-Box Testing

The tester does not have the code for the routine

- Tester only has a functional description of the routine
- Test inputs determined by requirements
- Techniques:
  - Graph-based
    - Data/Transaction Flow Modeling
    - Finite State Modeling
    - Timing Modeling
  - Equivalence Partitioning
  - Boundary Value Analysis
  - Comparison Testing (separate teams)

Equivalence Partitioning

Determine which classes of input data have common properties

- Philosophy:
  - If the program does not display erroneous output for one member of a class, then all members of that class should be “safe”.
- Example:
  - Input spec states the range of an input is a 5-digit number
  - Equivalence classes:
    - Values less than 10,000
    - Values between 10,000 - 99,999
    - Values greater than 99,999

White-Box Testing

- Tester has knowledge and access to the source code of the routine
- Does not need to understand the program as a whole, only the module being tested
- Hard to get clues about which test inputs best exercise the program
- Techniques: Control Structure Testing
  - Basis Path Testing
  - Condition Testing
  - Data Flow Testing

Basis Path Testing

- Derive a program flow graph
- Devise test cases that exercise each path of control flow
- Each decision, each loop, each statement is exercised
- Independent path: one which traverses at least one new edge in the flow graph.
- Maximum number of tests required for all conditions is the *Cyclomatic complexity* (McCabe, 1976)
  - Cyclomatic complexity (G) =Number of edges - Number of nodes + 2

Real-Time Systems Testing

Cannot use intrusive methods for testing

- printing, breakpoints etc.

- Reliability requirements are usually very high
- Depend on timing constraints
- Often interrupt driven
- Much interaction between processes
- Test each component individually in isolation
- Test threads - system reaction to an event
- Introduce multiple events
- Use emulators and simulators for devices

GUI Testing

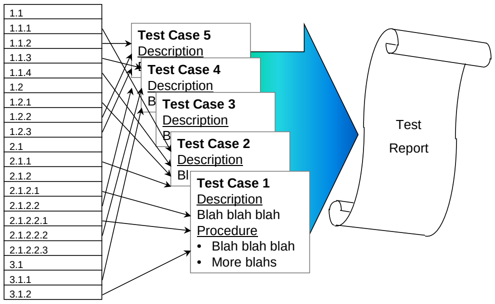
- Windows
  - window behavior
  - titles
  - data content
  - regeneration
  - highlighting when active
- Menus
  - menu bar related to context
  - addressability
  - functionality
  - help capability
- Data Entry
  - user comprehension
  - mechanical operation
  - data validation

Testability

A good software engineer designs a computer program, a system, or a product with testability in mind

- Operability: **“The better it works, the more efficiently it can be tested”**
- Observability: **“What you see is what you test”**
- Controllability: **“The better we can control the software, the more the testing can be automated and optimized”**
- Decomposability: **“By controlling the scope of testing, we can more quickly isolate problems and perform smarter re-testing”**
- Simplicity: **“The less there is to test, the more quickly we can test it”**

Test Plan Design



Document Summary

1 Introduction
- copy and paste directly from requirements specification

2 Requirements Identification
- copy and paste direct from requirements specification verification section

3 Test Plan / Procedures MORE LATER
- overall discussion of testing strategy – the idea is to develop as small a number of tests which cover all the requirements

4 Test Results
- table listing test scenarios, software version, results observed, signature of observer

5 Traceability Matrix
- copy and paste requirements list from section 2
- add column to show which scenario / event from the above list demonstrated each requirement

3 Test Plan / Procedures

- overall discussion of testing strategy
  - the idea is to develop as small a number of tests which cover all the requirements
- List of cases (1 is unusual but OK)
  - Case: [aka Scenario] single setup and sequence of events which build upon the results of the previous event

3.<n> Case: <name>

- Overall case (scenario) description

3.<n>.1 Event Description
- hardware, software and people required to run this test

3.<n>.1.1 Input Data
how to stimulate the required behavior

3.<n>.1.2 Events to Observe
what should happen, how to observe it

3.<n>.2 Event Description
- hardware, software and people required to run this test

3.<n>.2.1 Input Data
how to stimulate the required behavior

3.<n>.2.2 Events to Observe
what should happen, how to observe it

3.<n>.3 etc

Open Problems

- Despite the emergence of a formal approach to the definition of security requirements, computer security has gotten worse, not better over the last 30 years. (Karger and Schell, 1974 and 2002)
- K&S note that current secure systems are less secure than Multics, and “Multics, with its security enhancements, was only deemed suitable for processing in a relatively benign ‘closed’ environment.”
- What they see missing is a verifiable security kernel to block professional hacker attacks using malicious software trapdoors. This is a *currently emerging threat*.