Testing Software Quality Characteristics – Part 1

Configuration Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Utilize strategies for configuration testing

Multiple Configurations

- Should performance tests be repeated for each possible configuration?

Configuration Testing

Verify that the functional and performance requirements of the system are met for the different configurations that the system must run on.

Configuration Testing Steps

1. **Identify the parameters that define each configuration that could have an impact on the system's ability to meet its functional and performance requirements**
   - CPU
   - Operating system
   - Memory
   - Data Base
2. **Partition (group similar parameters to reduce possible number of configurations)**
3. **Identify configuration combination to test**
   - Boundaries (maximum and minimum
   - Risk based
   - Design of experiments pairwise combinations (used to select combinations of configuration parameters when testing all combinations is impractical or not needed)

Example: Performance Test of Car

Engine	Transmission	2D/4D	Tires
3.0	auto (a)	2D	15 inch normal (15 n)
3.8	manual (m)	4D	15 inch high performance (15 hp)
5.0			

There are 24 configuration combinations.

Performance Test of Car (continued)

Slowest Configuration

3.0 / auto / 4D / 15 n

Fastest Configuration

5.0 / manual / 2D / 15 hp

Risk-based Selection Based on Projected Sales

3.8 / auto / 4D / 15 n

Pairwise Combinations

Engine	Transmission	2D/4D	Tires
3.0	a	2D	15 n
3.0	m	4D	15 hp
3.8	a	2D	15 hp
3.8	m	4D	15 n
5.0	a	4D	15 hp
5.0	m	2D	15 n

Summary

Testing Software Quality Characteristics – Part 1

Performance Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Utilize strategies for performance testing

Performance Testing

Objective

Verify that the system meets its performance requirements for specified load conditions including stress and volume scenarios.

Entry Criteria for Performance Testing

- **Quantitative and measurable performance requirements**
- **Reasonably stable system**
- **Test environment representative of customer site**
- **Tools**
  - Load generator
  - Resource monitor

Test Data Management

- Growing complexity and volume of data in today's applications requires emphasis on test data management
- Full data sets replicating real data from operational system is ideal

Exercise

- **Assume you are a system tester for a new airline reservation system.**
- **Discuss how to improve the following performance requirement:**
  - "The airline reservation system shall have excellent response time."

Load Specification

For relevant use-cases it is important to specify performance requirements in terms of load

Load may reflect:

- Different volumes of activity
  - Busy hour in a cell phone system
  - Same call profile used
- Different mixes of activity
  - Earthquake or tornado in a cell phone area
  - Different call profile

Varying Load During Performance Testing

- **For relevant use-cases the load should be varied and response time tracked**
  - Verify performance requirments
- **Resource usage can also be tracked as the load is varied to identify potential bottlenecks or sources of performance problems**

Summary

Testing Software Quality Characteristics – Part 1

Regression Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Utilize various strategies for regression testing

Regression Testing

- **Modifying existing software is a high risk-activity**
- **Modifications occur due to:**
  - Error fixes
  - Incorporation of new functions
- **Modifications may introduce errors due to:**
  - Code ripple effects
  - Unintended feature interactions
  - Changes in performance, synchronization, resource sharing, etc.

Examples

GTE / Arizona State University study found that almost half of problems detected during system test of a large switching system were in features that worked fine in the previous release

Caper Jones study found bad fix injection rates vary from less than 2% to more than 20%

Ariane 5 rocket controller

- $350 million dollar loss
- Didn't regression test code
- Assumed dynamics of rocket were the same as Ariane 4

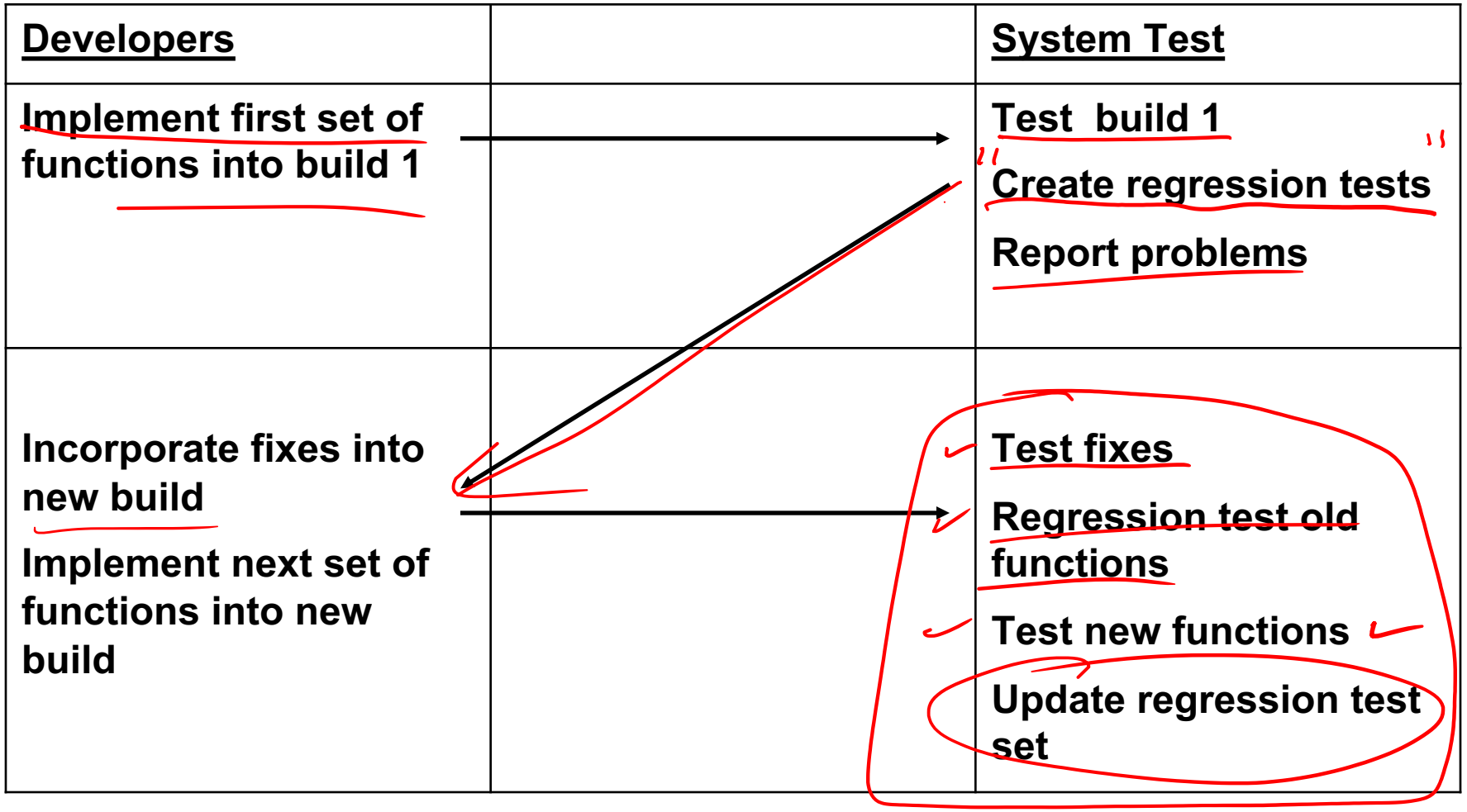
Objective

Objective of regression testing is to ensure that previously developed and tested functions continue to work as specified after software modifications have been made

Regression testing must be approached from a multi-level point of view

- Unit level regression tests
- Integration level regression tests
- System level regression tests

Incremental Development and Testing Process Overview



Detailed Defect Repair Process

System Test

Execute test and detect error

Write problem report.

Problem Analyst

Analyze problem and recommend corrective action

Development

repair code.

Inspect change.

Unit test change.

Change Control Board

Decide to repair, cancel or defer

Regression test components modified.

Add changes to next build.

Test fix.

Regression test.

Regression Test Strategies

Full

- Rerun all existing tests in response to a code modification
- Normally impractical

Selective

- Rerun a selected subset of tests based on the modification
- Execute a standard confidence test irregardless of the modification

Selective Regression Testing Based on Modification

- **Typically requires tools and close communication with developer**
- **Strategies include:**
  - Testing of code deltas
  - Ripple effect analysis

Testing of Code Deltas

- **Requires coverage tool for mapping test cases to code at desired granularity level:**
  - Code block
  - Component
- **Requires configuration management tool to identify code change deltas**
- **Strategy suggests re-running tests that traversed changed or deleted code**

Example

Tests

Components

	T1	T2	T3	T4	T5
A	X	X	X	X	X
B	X		X		
C		X			X
D	X			X	
E			X		
F		X			X

If components B and D are changed, tests T1, T3 and T4 are re-executed.

Ripple Effect Analysis

- Requires developers to identify the impact of changes on other requirements or features
- Best addressed via checklist items in a modification inspection
- Potentially impacted requirements and features are communicated to system test

Example

Tests

Components

	T1	T2	T3	T4	T5
F1	X				
F2		X			X
F3			X		X
F4				X	
F5	X				X
F6		X			
F7				X	

If a modification to F1 is determined to impact F3, then tests T1, T3 and T5 are re-executed.

Selective Regression Testing Using a Confidence Test Suite

- Select a subset of tests to execute to verify previous functionality
- Include tests addressing:
  - High frequency use-cases
  - Critical functionality
  - Functional breadth
- Inspection of test cases should include a checklist item addressing the suitability of particular tests for the confidence test suite

Revalidation Issue

- Regression tests must be revalidated to ensure they are consistent with the software modification
- Test inputs and expected outputs must be re-examined for correctness

Summary

Testing Software Quality Characteristics – Part 1

Stress Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Utilize strategies
for stress testing

Stress Testing

- Verify the behavior of the system meets its requirements when its resources are saturated and pushed beyond their limits.
- Attempt to find the stress points and ensure the system performs as specified

Analogies

- Human stress test
- Automobile stress test
- Bridge stress test

Stress Testing Steps

1. **Identify stress points**
   - Work with developers and architects
   - Identify potential bottlenecks
2. **Develop a strategy to stress the system at points identified in Step 1**
   - Often requires load generation tools
3. **Verify that intended stress is actually generated**
   - Stress testing strategy may be ineffective
   - System performance may be better than expected
4. **Observe behavior**
   - Verify stress related requirements are met
   - Ensure functional correctness

Summary

Testing Software Quality Characteristics – Part 1

Volume Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Utilize strategies
for volume testing

Volume Testing

Verify the behavior of the system meets its requirements when the system is subjected to a large volume of activity over an extended period of time.

Errors Targeted by Volume Testing

- Memory leaks
- Counter overflow
- Resource depletion

Summary

Testing Software Quality Characteristics – Part 2

Error Detection, Recovery and Serviceability Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Develop error detection, recovery and serviceability tests

Error Detection and Recovery Testing

- **Overall system reliability and availability is dependent upon the system's ability to detect and recover from a variety of failures**
  - User
  - Hardware
  - Software
  - Other systems

Error Detection and Recovery Testing (cont'd)

- It is essential to have a list of the errors to recover from specified in the requirements
- Usual testing approach consists of error injection

Serviceability Testing

- **Important for system availability**
- **Objective is to verify serviceability requirements are being met**
  - e.g. "critical problems will receive fixes or workarounds within 4 hours"
- **Serviceability includes all aspects of problem reporting, isolation, correction, verification and fix release**
- **Usual testing approach is to inject a failure and assess response**

Summary

Testing Software Quality Characteristics – Part 2

Reliability Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Apply operational profile testing to assess software reliability

Reliability Definition (from John Musa)

- “The probability that a system or a capability functions without failure for a specified time or number of natural units in a specified environment"

- Natural units correspond to the processing performed such as the number of calls or transactions completed (e.g. one transaction lost per 50,000)

- Probabilities have a range of 0 to 1

Availability Definition from John Musa

> “The probability at any given time that a system or capability of a system functions satisfactorily in a specified environment"

Software Availability Calculation

- **Availability can also be performed as the percent of time the system performs satisfactorily**

$$\text{availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \times 100\%$$

5NINES Availability Requirement

- **The system shall be available 99.999% of the time, i.e. the probability the system functions satisfactorily at any point in time is 99.999**

- **In terms of service to a customer over a year, this translates into approximately 5 minutes of allowable down time per year**
  - There are 525,600 minutes per year
  - 5NINES implies one minute of down time per 100,000 minutes

Achieving High Reliability and Availability

Achieving high reliability and availability in a cost effective manner requires the prudent application of Software Reliability Engineering techniques

Appropriate development techniques must be applied for:

- Fault prevention
- Fault tolerance

Achieving High Reliability and Availability (cont'd)

Appropriate testing techniques must be applied along with models for assessing whether reliability and availability objectives are being met.

- Operational profile testing
- Error detection and recovery testing
- Serviceability testing

Introduction to Operational Profiles

- An operational profile describes how users utilize a product
- An operational profile consists of a set of major functions performed by the system and their occurrence probabilities
- An operational profile is essential for reliability prediction

Basic Operational Profile Construction Steps

1. **Identify the major functions performed by the system**
   - Identify different types of users / external entities
   - Use-cases are good candidates for basing the operational profile
2. **Identify the occurrence rates**
   - Historical data
   - Marketing
3. **Calculate the occurrence probability**

ATM Example

Use-Cases	Occurrence Rate (xact / hr)	Occurrence Probability
Deposit	95	0.095
Withdraw	900	0.9
Transfer	5	0.005

Development of Tests

- Tests are developed based on the operational profile
- Test generation is modified to incorporate critical functions with low occurrence probabilities
- Number of tests to execute is based on the reliability objectives

Interpreting Failure Data

Development Testing

- Goal: To remove faults that have causes failures

Certification Testing

- Goal: To determine whether a software component or system should be accepted or rejected

Other Uses of Operational Profiles

Operational profiles may also be used to:

- Guide development priorities
- Assist in performance analysis

Summary

Testing Software Quality Characteristics – Part 2

Reliability Models

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Identify how software reliability models work

Modeling Software Reliability Growth

- Reliability growth model shows how reliability changes over time
- Models support answering "when to stop testing?" question
- Numerous models exist
- Effectiveness of any reliability measurement is directly related to the effectiveness of collecting the right data during testing such as:
  - Failure intensity: the number of failures per natural or time unit

Time to Failure

Cumulative Number of Failures

Number of Failures Per Unit of Time

Software Reliability Models

- **All models possess assumptions such as:**
  - No new errors are introduced by fixes
- **Models generally utilize a mathematical distribution to represent reliability growth**
  - Poisson
  - Exponential
- **Effective model predictions require testing with an operational profile**

Statistical Testing

- Testing software for reliability rather than fault detection
- An acceptable level of reliability should be specified and the software tested and amended
  until that level of reliability is reached

Reliability Testing Problems

Operational profile uncertainty

- Is the operational profile an accurate reflection of the real use of the system

High costs of test data generation

- Very expensive to generate and check the large number of test cases that are required

Statistical uncertainty for high-reliability systems

- It may be impossible to generate enough failures to draw statistically valid conclusions

Growth Model Selection

- Many different reliability growth models have been proposed
- No universally applicable growth model
- Reliability should be measured and observed data should be fitted to several models
- Best-fit model should be used for reliability prediction

Summary

Testing Software Quality Characteristics – Part 2

Security Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Identify basic security testing approaches

Security Testing

- Software correctness and security are not the same
- Most applications contain private data
- Goal of security testing is to ensure private data is protected from unauthorized users

Security Fundamentals

Confidentiality

- Application
- Data

Integrity

- Data modification
- Functions performed

Availability

- Denial of service

Security Testing Context

- Software may have unintended or unknown functionality that may produce side-effects contributing to security problems
- Security flaws require testing software interactions with its environment

Components that Might Exploit Software

- OS
- File System
- GUI
- Other systems (databases, libraries, etc.)

GUI Security Risks

Verify access control

- Entry to system
- Access to functions and data

Look for all possible access methods to data

- Cut and paste
- Screen capture

Evaluate malicious input

- Denial of service

File System Security Risks

- Evaluate how data is stored and retrieved
- Focus on encryption and data protection

OS Security Risks

Evaluate decrypted data storage in memory

Stress test with low memory

- System under memory stress may leave data unprotected

Other Component Security Risks

- Consider results of component failure
- Components may consist of libraries, databases, etc.

Security Testing Strategies

- **Deny application access to libraries it needs**
  - Ensure crashes do not impact security
- **Try to overflow input buffers by inputting long strings**
- **Try special characters as inputs**
- **Try default or common user names and passwords**

Security Testing Strategies (cont'd)

Attempt to fake the source of data

- Consider a system with packets sent over the network which contain source identifier
- Fake source in packet

Force system to use default values

- Do not enter data when prompted
- Exploit time outs

Security Testing Strategies (cont'd)

- **Test all routes to perform a task**
  - Consider opening a file
  - Ensure all scenarios go through security validation
- **Produce each error message and ensure that it does not compromise security**

Approaches for Improving Security Testing

Consult public security databases

- CERT (www.cert.org)
- Contain information about published software bugs

Reason about errors in databases and possible vulnerabilities in your product

- What caused the failure
- How might it have been detected during test
- Is system vulnerable to attack

Summary

Testing Software Quality Characteristics – Part 1

Usability Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Generate usability tests

Usability Testing

- Verify the behavior of the system meets its requirements when its resources are saturated and pushed beyond their limits.
- Attempt to find the stress points and ensure the system performs as specified

Usability Testing

Close to one-half of code in many applications is in the user interface

Usability is the degree to which intended users are:

- Able to perform tasks the product is intended to support in intended environment
- Satisfied by the procedures they must follow and the resultant output
- Protected from consequences of their actions

Usability Requirements

Usability requirements are typically stated in terms of:

- **Learnability**: the type and amount of training required to bring users to a desired level of performance
- **Memorability**: addresses the ability to retain skills in using a product once it is learned
- **Errors**: measures the number of incorrect actions a user makes in trying to accomplish a task
- **Efficiency**: measures the speed with which tasks can be performed
- **Subjective satisfaction**: the user's overall feeling about the product

Usability Testing Reliability and Validity Concerns

Reliability: would you get the same results if test were repeated

- Best user is 10X faster than slower
- Best 25% are 2X faster than

Validity: does usability test measure something of relevance

- Wrong users
- Wrong task

Test Goals

Formative Evaluation

- Learn which aspects of interface are good and bad
- How can design be improved

Summative evaluation

- Assess the overall quality of the interface
- Measurement test

Test Plan Concerns

- Who are the users?
- What task will they perform?
- What user aids will be available?
- What data is to be collected?
- What criteria will be used to determine success?

Pilot Tests

Test procedures must be tried out in a pilot study

Evaluate

- Instructions
- Success criteria
- Time to perform tasks
- Evaluation criteria

Identifying Test Users

- Users must be representative
- Evaluate with both novice and expert users
- Be prepared to train users to achieve expert level

Usability Comparison

- **When evaluating usability choices care must be taken when using the following testing strategies:**
  - Between subject testing
  - Within subject testing
- **Within subject testing is preferable**

Ethical Aspects with Human Subjects

- Subjects may have concerns about performing inadequately
- Need to make subjects feel comfortable
- Emphasize system is being tested and not the user
- Maintain privacy issues

Test Tasks

- Must be representative
- Begin with easy tasks to boost confidence
- Give tasks one at a time

Stages of Test

- Preparation (ensure environment is set-up)
- Introduction (welcome, purpose, overview)
- Running the test
- Debriefing

Thinking Aloud

- Test subject uses system which continuously thinking out loud
- Testers may need to periodically prompt test subject

Usability Lab

Two-way mirror

Video cameras

- User faces
- How user is interacting with system, doc, etc.

Summary

Test Management Part 1

Risk Based Testing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Perform risk based testing

Risk-Based Testing Strategy

- **Applicable when project constraints make it necessary to prioritize testing**
- **High risk areas are identified as a function of the:**
  - Likelihood of a failure occurring
  - Severity of failure should it occur
- **Can be applied at various levels of abstraction:**
  - Subsystem
  - Feature
  - Component

Assessing the Likelihood of a Failure

- **Errors cluster**
- **Some areas of the system may be more error-prone due to:**
  - Complexity
  - New or changed code
  - Outsourced development
  - Poor history

Assessing the Consequences of a Software Failure

- Requires interaction with customers and developers
- Failures of capabilities can be assessed in terms of severity
- Severity must address issues such as system:
  - Reliability
  - Availability
  - Performance
  - Usability
  - Compatibility
  - Maintainability
- Requirements / use-cases should be prioritized to support risk-based testing

Assessing the Consequences of a Software Failure

- For more complex systems, consequences of a component or function failure may not be obvious in terms of its severity
- Rigorous failure analysis can be performed with assistance of developers using techniques such as:
  - Fault trees
  - Failure mode effect analysis

Risk-Based Testing Strategy

Summary

Test Management Part 1

Test Documentation

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Identify the different types of test documentation

IEEE Test Documents

See IEEE STD-829 IEEE Standard for Software Test Documentation

Templates are provided for:

- Test plan
- Test case
- Test Incident Report
- Test Summary Report
- Other test documentation

Test Case

- Test case specification identifier
- Test items
  - References to documents used to produce test
- Input specifications
- Output specifications
- Environment for executing tests
- Special test procedures to follow
- Intercase dependencies

Test Incident Report

Test incident report identifier

Summary

Incident description

- Inputs
- Expected results
- Actual results
- Date and time
- Environment
- Attempts to repeat

Impact

Test Incident Reports

- **Must be carefully written**
- **Target audience includes:**
  - Programmers
  - Other testers
  - Managers
  - Quality Assurance
  - CCC Board

- **Must provide enough information for each viewer to do their job**

Developer Mentality

Testers must understand what it is like to work as a programmer

- Complex tasks
- Vague and changing requirements

Most developers are specialized in a part of the system

- Testers are generalists
- Testers must help developers understand the big picture

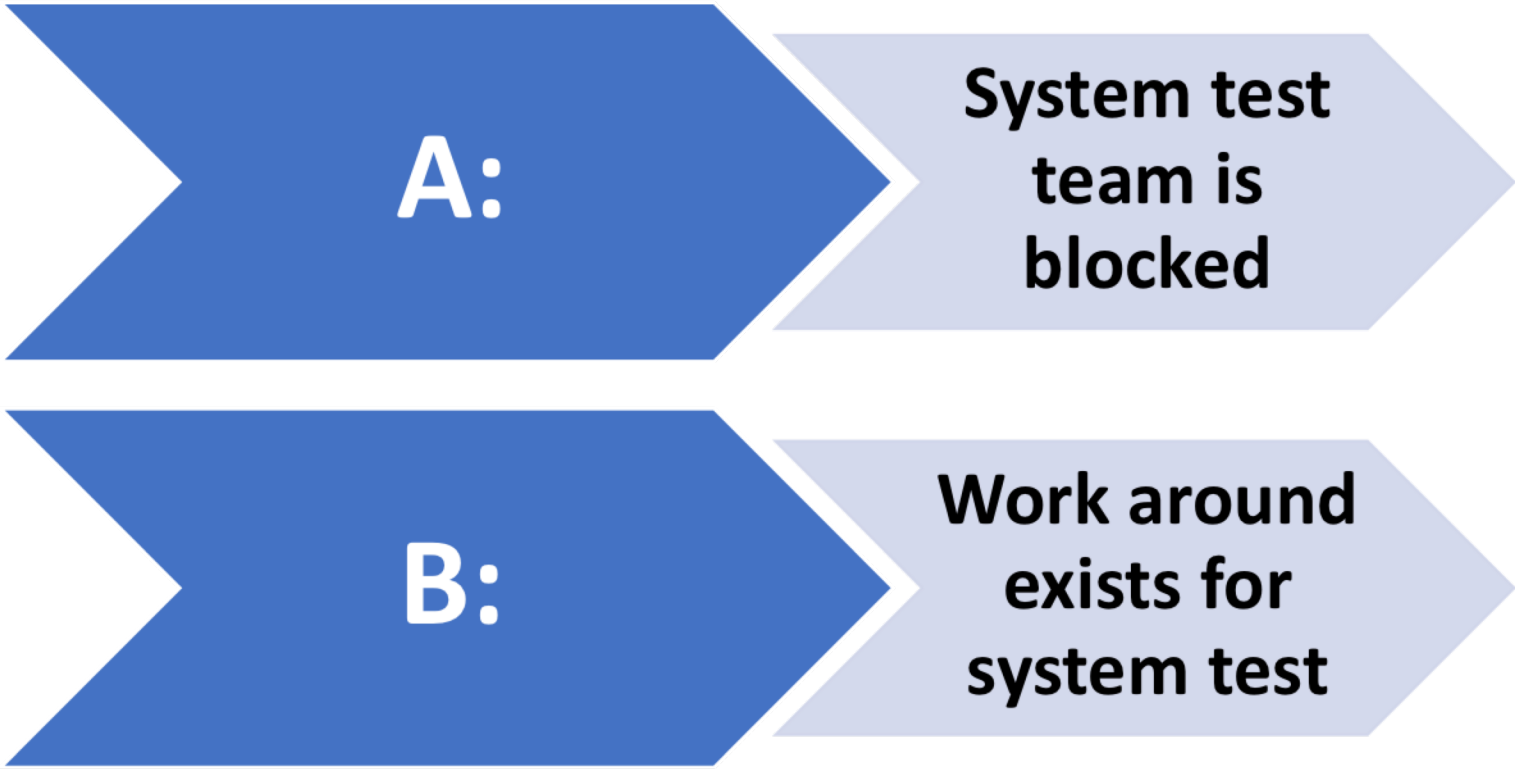
"context"

Severity vs Priority

- Testers need to recognize the difference between severity and priority
- Severity reflects the customer impact of the error
- Priority reflects project considerations

- Not all high severity errors are high priority
- Not all high priority errors are high severity

System Test Problem Priorities



Review Defect Reports?

To save time, some organizations have bug reports reviewed by another tester

Review criteria include:

- Completeness
- Repeatability
- Clarity
- Severity evaluation

Test Summary Report

Test Documentation

Agile development methods have forced rethinking of documentation requirements

Level and formality of test documentation must be based on:

- How will documentation support the testing activity
- Is documentation a deliverable
- Are their regulatory concerns
- Will documentation support tracking activities
- Support for regression testing

Summary

Test Management Part 1

Test Estimation Steps

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Follow a process
for estimating
testing efforts

Importance of Estimates

Estimates are the foundation for:

- Testing budget
- Allocation of staff and resources

Testing estimates are often made as a part of the overall development project estimate

Major Causes of Inaccurate Estimates

- Misunderstanding of requirements
- Overlooked tasks
- Insufficient analysis when developing estimates due to time pressure
- Lack of guidelines for estimating
- Lack of historical data
- Pressure to reduce estimates

Generic Estimation Process

1. Determine estimation responsibilities
2. Review and clarify testing objectives, deliverables, milestones and constraints
3. Identify testing tasks
4. Select appropriate size measure for testing work
5. Select size estimation method
6. Estimate and document size
7. Estimate and document effort

Identification of Testing Tasks

- **A set of testing activities must be defined**
- **Testing tasks include:**
  - Understanding requirements
  - Training on tools
  - Test case development

Select Appropriate Size Measure

Size of the testing activity must be estimated utilizing some type of standard component

Possible testing size measures include:

- Number of requirements
- Number of use cases / scenarios
- Lines of code to be tested

Select Size Estimation Method

Top-Down / Analogy

Bottom-up

Top-Down / Analogy

- An expert develops an estimate based on past similar projects
- Does not work well for new types of projects
- May fail if estimator does not take into account differences between projects such as expertise, difficulty of testing, stability of requirements, etc.

Bottom-Up Estimation

- **Breaks the testing effort into parts**
- **Parts may consist of:**
  - Functions to test
  - Types of testing to perform
- **Each part is estimated separately and the parts are summed to create the estimate.**
- **A work breakdown structure may be developed to organize the parts**

Estimate and Document Size

- 80/20 rule applies to size estimation
- Accuracy of the size estimate must also be communicated
- Prior to effort estimation, the size of work must be estimated and documented along with the basis of the estimate and the assumptions made

Estimate and Document Project Effort

- **A size unit for effort must be selected**
  - Staff hours
  - Staff months
- **actual effort is a function of:**
  - Size estimates
  - Estimated productivity

- **Effort estimate must be documented along with the basis for the estimate and the assumptions made**

Strategies for Checking Estimates

- Check that estimates for similar parts of the system are the same
- Check resource estimates of parts of your system with similar parts of other completed systems
- Review with other experts

Summary

Test Management Part 1

Test Estimation

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Estimate testing effort

Estimating System Testing Time

House Cleaning Analogy

Developing Estimates

Once the variables to take into account in developing the testing estimate have been identified, an estimate based on them must be produced

Approaches for developing estimates based on variables include:

- Utilization of historical data
- Calculation via a cost estimation model
- Generating a test estimate based on a percentage of development estimate

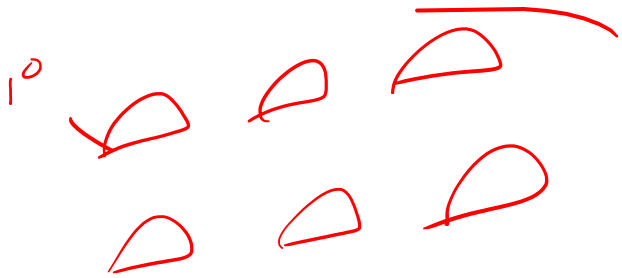
Summary

Test Management Part 1

Test Exit Criteria

Ira A. Fulton Schools of Engineering
Arizona State University

Objective



Objective

Select test exit criteria

When to stop testing?

1. Out of time / $
2. coverage
3. Reliability models
4. curves / graphs
5. analytics found all defects
6. customer is satisfied
7. test objectives

System Test Exit Criteria

The best criteria for stopping testing is when the test objectives have been met

Typical approaches for determining that the system is ready for release include:

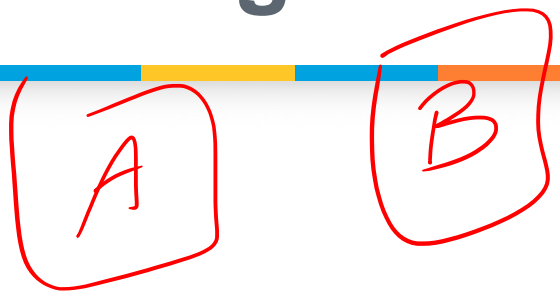
- Measuring defect density
- Defect pooling
- Defect seeding
- Trend analysis
- Reliability modeling

Measuring Defect Density

- **Defect density is defined as number of defects per thousand lines of code**
- **If available, historical defect density data might be used to predict expected number of defects to be found in system test**
- **Actual number might be compared against expected number to determine if system is ready for release**

- **Problems with this approach include:**
  - Lack of historical data
  - Defect injection rate for current system might be different than historical data
  - Defect containment for previous phases might be different than historical data
  - Defects found may not impact customer's perception of the system

Defect Pooling



- Appropriate to apply when operational usage scenarios are being executed
- Requires defect reports to be broken into 2 groups which can be tracked separately
- Assumes that each group of defects reflects independent testing of whole system based on operational usage
- Can be implemented many ways such as:
  - Splitting the system testers into 2 groups
  - Collecting defect data from 2 independent beta test groups

Defect Pooling Approach

Calculate the following assuming:

- $Defects_A$: is the number of defects found by Group A
- $Defects_B$: is the number of defects found by Group B
- $Defects_{A+B}$ is the number of defects found by both Group A and B

Unique Defects =
$(Defects_A + Defects_B) - Defects_{A+B}$

Estimated Total Defects =
$(Defects_A \times Defects_B) / Defects_{A+B}$

Estimated Remaining Defects =
Estimated Total Defects - Unique Defects

Defect Pooling Example

Assume:

- Group A found 30 defects
- Group B found 40 defects
- 20 defects were common to both Group A and Group B

$Defects_A = 30$ $Defects_B = 40$ $Defects_{A+B} = 20$

Unique Defects = (30 + 40) - 20 = 50

Estimated Total Defects = (30 x 40) / 20 = 60

Estimated Remaining Defects = 60 - 50 = 10

Defect Seeding

- Controversial approach involving "seeding" defects into the system
- Assumes that defects can be inserted into the system that are representative of defects that customers will encounter
- Assumes ability to detect remaining defects is equivalent to ability to detect seeded defects
  - Estimated Total Defects = (seeded defects planted / seeded defects found) x (normal defects found)

Defect Seeding Example

Assume:

- 20 defects are seeded
- 10 seeded defects are found by test
- 50 additional defects are found by test

Estimated Total Defects = (20 / 10) x 50 = 100

Estimated Remaining Defects = 100 - 50 = 50

Trend Analysis

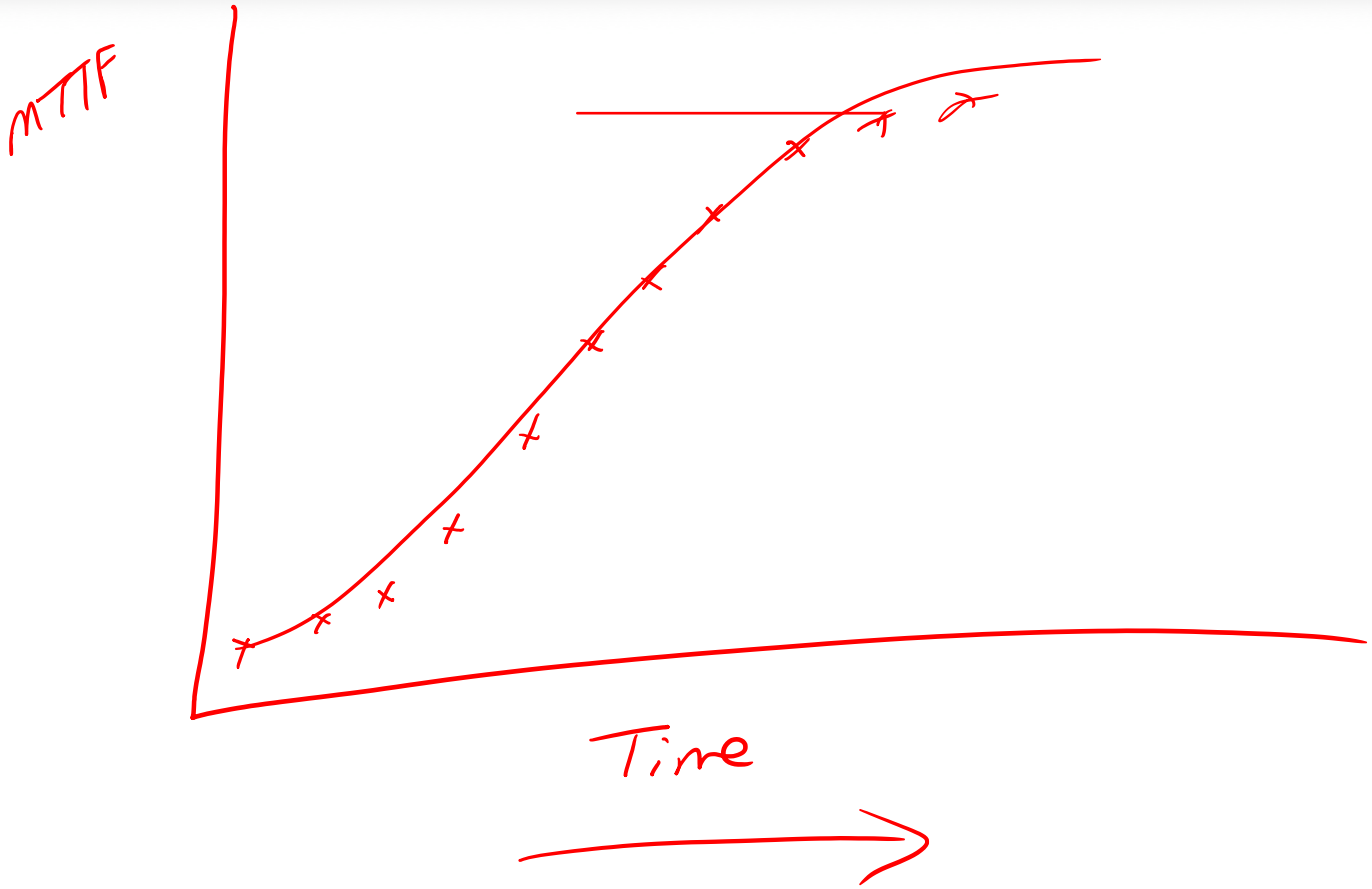
Software readiness may be assessed via analysis of trend data:

- Time to failure
- Cumulative number of failures
- Number of failures per unit of time (failure intensity)

One of three trends can be identified:

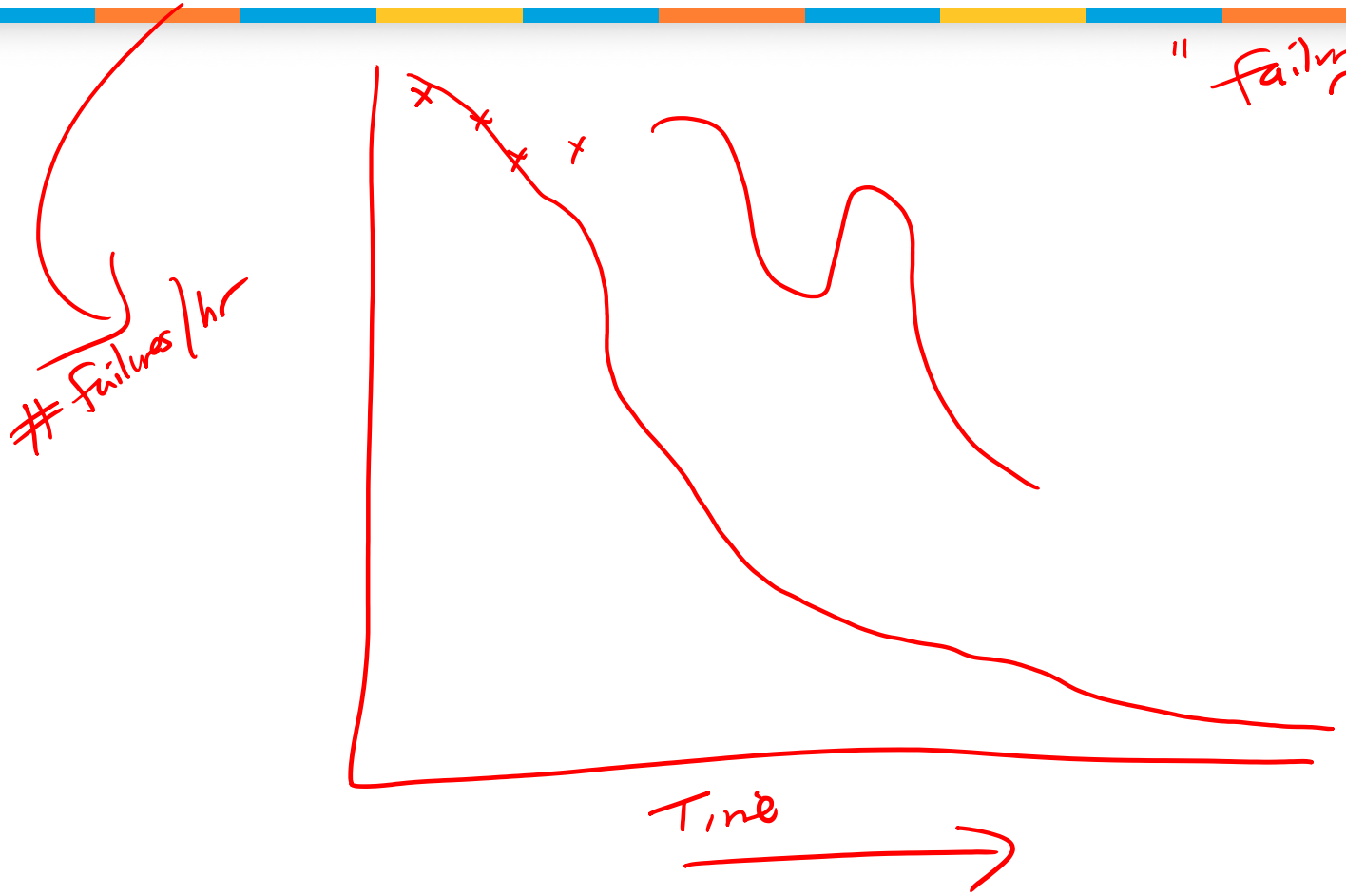
- Decreasing reliability
- Increasing reliability
- Stable reliability

Time to Failure



Cumulative Number of Failures

Number of Failures Per Unit of Time



Trend Analysis (Decreasing Reliability)

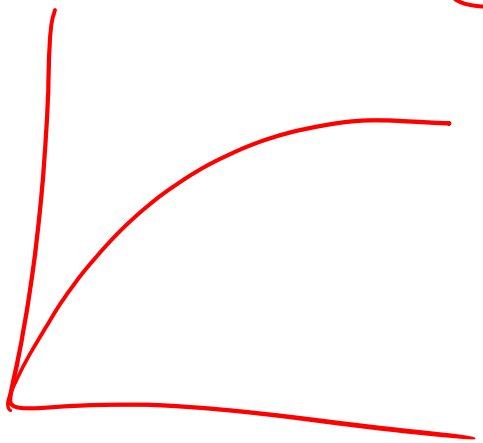
Expected at the start of a new activity

- New testing phase
- New type of testing
- Different user profile

Long duration signals significant software problems

Trend Analysis (Increasing Reliability)

- Normally good news



- Sudden increase may, however, be due to:
  - Changing test effort
  - Test burnout
  - Unrecorded failures

Summary

Test Management Part 1

Test Planning Overview

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Identify the major components of a system test plan

Test Planning

Test plans should be written for all testing levels:

- Unit
- Integration
- System
- Beta
- Acceptance

System Test Plan

- A well-thought out system test plan is essential to success of a testing effort
- System test plan must reflect an in-depth understanding of the objective of the system test as well as project constraints
- System test planning must begin early (during the software development requirements phase)

System Test Plan (continued)

The system test plan must address:

- System test objectives
- Dependencies and assumptions
- Adopted test strategy
- Specification of the test environment
- Specification of system test entry and exit criteria
- Schedule
- Risk management

System Test Objectives

- The system test plan must clearly define the objectives of the system test activity
- Possible objectives were presented in Unit 1

Dependencies and Assumptions

When creating the system test plan, all dependencies and assumptions must be identified

Examples include:

- Resource availability
- Software completed on time
- _____
- _____

Testing Strategy

Testing strategy defines how testing objectives will be met within project constraints

Testing strategy determines:

- Techniques to be used for test data generation
- Test environment
- Entry and exit criteria
- Schedule

Testing strategy is often risk-based

Specification of the Test Environment

Test environment issues include:

- Platforms to test on
- Simulators
- Testing tools

Selection of test environment is based on objective of testing and test strategy chosen

- Performance testing objective may require load generation tools
- Configuration testing objective may require additional resources and/or simulation tools

System Test Entry Criteria

- **Established based on test strategy to maximize test effectiveness**
- **Problems with beginning system test too early include:**
  - Inability to run all tests
  - Excessive communication with developers on problem fixes
  - High degree of retest
- **Possible entry criteria include :**
  - Code under configuration management
  - Completion of integration test
  - No outstanding high priority problems
  - Successful completion of system test readiness assessment

System Test Readiness Assessment

- Developed early in the project in conjunction with development
- Identifies functions and code stability needed to effectively begin system test
- Provides a concrete entry criteria for system test
- Provides a way for development to prioritize their activities as the start of system test grows near

Creation of System Test Schedule

Creation of a testing schedule requires the following activities:

- Identify all of the testing tasks to be performed
- Identify dependencies among the testing tasks
- Estimate the effort and resources needed to perform each task
- Assign tasks to individuals or groups
- Map testing tasks to a time line

Test Plan Risk Management

- System testing risks correspond to scenarios that could impact testing schedule or effectiveness
- Testing risks can be identified via checklists or previous project "lessons learned”
- Testing risks must be prioritized and mitigated
- Prioritization is based on likelihood of risk occurring and consequences
- Risk mitigation involves reducing the likelihood of the risk occurring and/or developing contingency plans to minimize impact of the risk should it occur

Summary

Test Management Part 1

Test Schedule

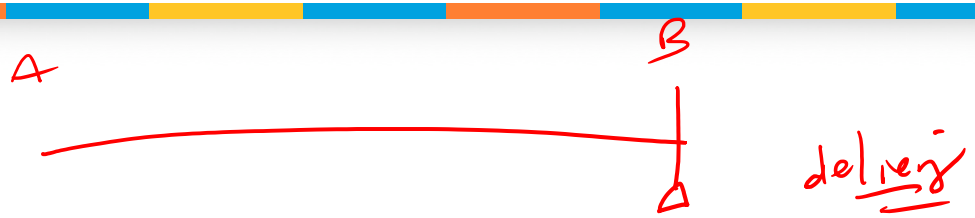
Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Create a test schedule

Creation of System Test Schedule



Creation of a testing schedule requires the following activities:

- Identify all of the testing tasks to be performed
- Identify dependencies among the testing tasks
- Estimate the effort and resources needed to perform each task
- Assign tasks to individuals or groups
- Map testing tasks to a time line

Testing Tasks Examples

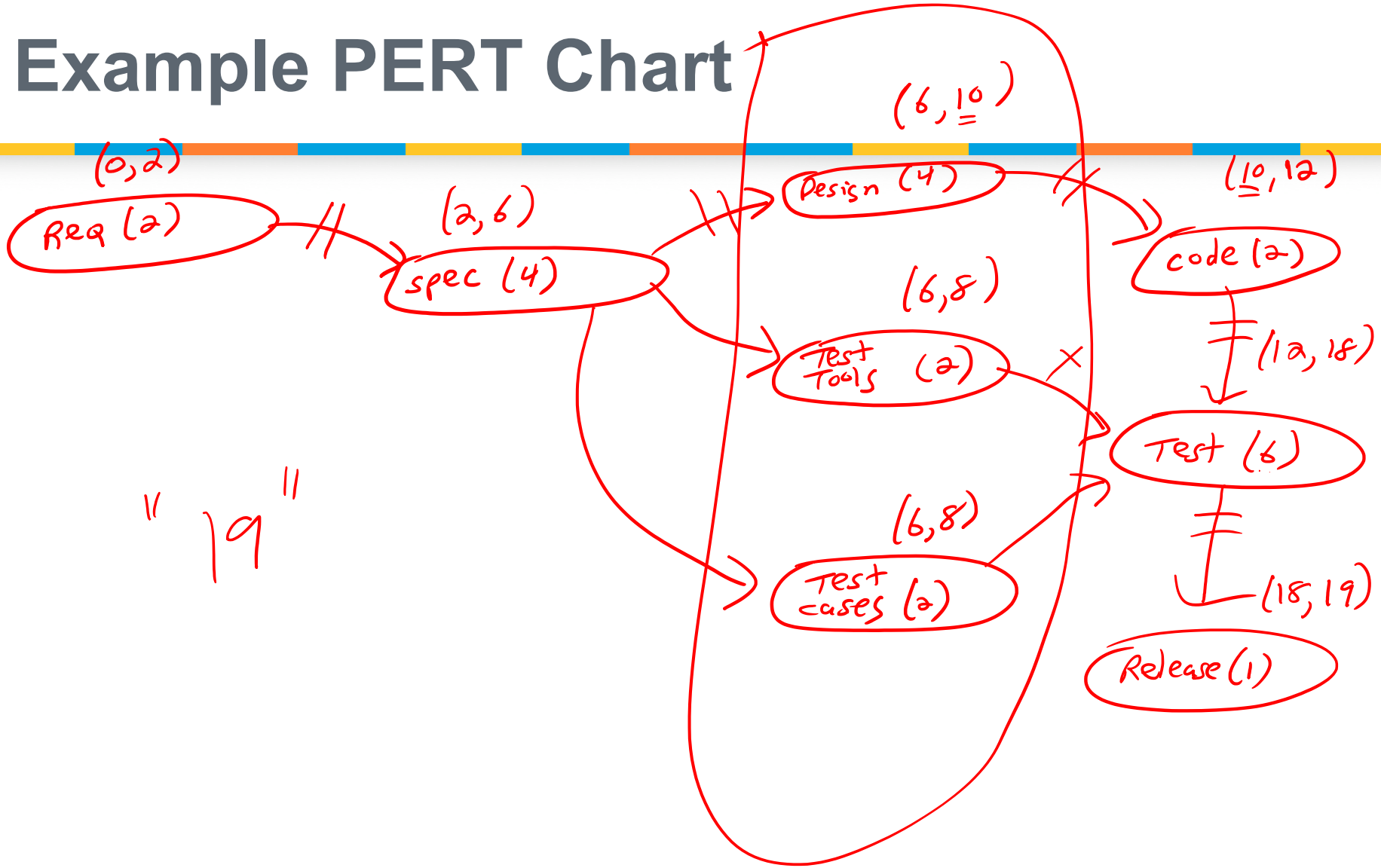
- Develop test plan
- Understand requirements
- Develop tests
- Review tests
- Install test environment
- _____
- _____
- _____

Analyze Dependencies

- After tasks are identified, a dependency analysis among tasks must be performed
- Dependencies can be documented in a PERT Chart
- PERT (Program Evaluation and Review Technique) was developed by the Navy in the 1950's

- Chart is a network whose nodes represent project activities and their associated duration's (often calendar weeks) and whose links represent precedence relations

Example PERT Chart



Critical Path Analysis

- What is the minimum time it will take to complete the project?
- What activities are critical to being able to complete the project in minimum time?
- What activities can be done in parallel?
- How long can each activity be delayed before it affects the finish date?

Critical Path Identification

- Critical path(s) is the path through the PERT chart with no slack time
- Can be identified by associating with each node, its earliest start and finish time
- Those paths where the earliest start time is always equal to the predecessor's nodes earliest finish time correspond to critical paths

Estimating System Testing Time

- A critical part of test planning is estimating the time needed to meet the testing objectives
- Overestimates lead to inefficient testing and delayed product release
- Underestimates lead to lots of overtime, high stress and probable ineffective testing

Assign Task Responsibilities

- Assign similar tasks to the same person
- Minimize necessary communication
- Match knowledge and skills to the task

- Assign tasks to people so that they learn and grow
- Attempt to accommodate preferences

Map Tasks to a Time Line

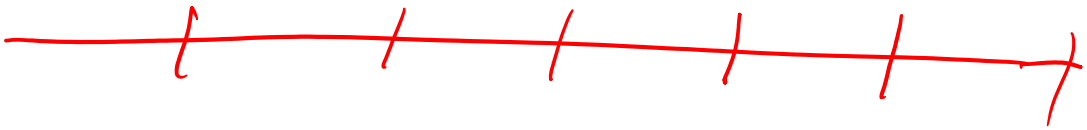
Schedule must be well thought out and take into account:

- Constraints
- Task dependencies
- Availability of personnel
- Risks

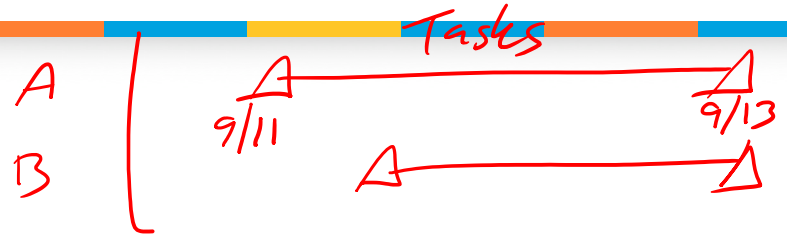
Developing a schedule is an iterative process:

- Adjust tasks, durations, resources and sequencing

Participants must commit to the schedule



Gantt Chart



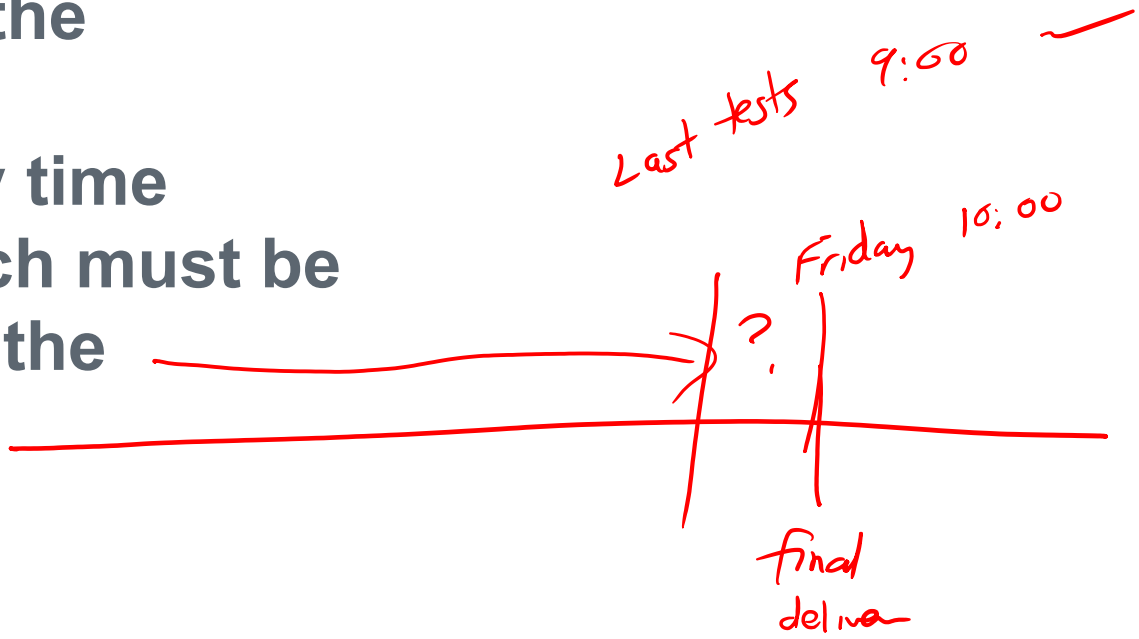
- Schedule can be documented in a Gantt chart
- Gantt chart identifies duration of tasks along with their starting and ending dates
- Gantt charts identify parallel tasks
- Multiple Gantt charts can be developed to show various levels of detail
  - Hierarchy of Gantt charts

Example Gantt Chart

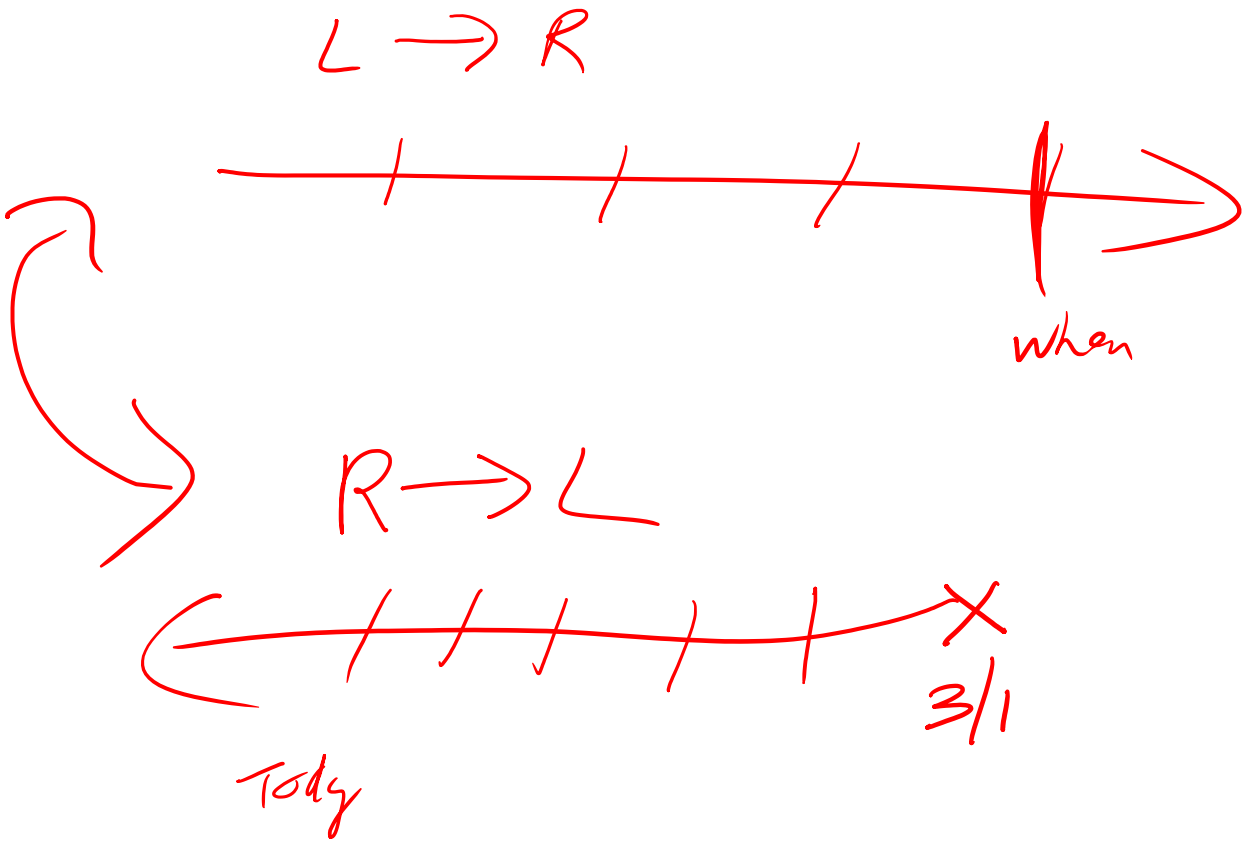
Schedule Buffers

- Risk management must be used to guide the team in the amount of contingency time (buffer) which must be allocated to the schedule

- Schedule confidence is tied to the buffer



Summary



Test Management Part 2

Causal Analysis

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Apply Causal Analysis to improve test effectiveness

Causal Analysis

- **First introduced by IBM in 1983**
  - IBM reports an investment of less than 1% in causal analysis achieved a 50% reduction in error rates
- **Seeks to identify the root cause of defects as well as approaches to eliminate future occurrences**
- **Defect data is collected and analyzed**

Causal Analysis Steps

Select a set of defects missed by testing to analyze

For each defect identify the probable cause it was not detected

Identify common causes among defects

Identify possible solutions to eliminate the common defect causes

Defect Cause Categories

Communication Failure

Defect was missed due to missing or incorrect information

Examples:

- Problems with requirements doc
- Last minute requirements changes not communicated to test team

Defect Cause Categories (cont'd)

Oversight

Defect was missed because of a failure to consider all cases or combinations when testing

Examples:

- Failing to test a feature interaction
- All combinations of inputs not addressed
- Boundary value not tested

Defect Cause Categories (cont'd)

Education

Defect was missed because of a lack of understanding of testing methodologies or tools

Examples:

- Design of experiments was incorrectly applied
- Tester didn't understand how automated tests were generated

Defect Cause Categories (cont'd)

Transcription

Missed defect was a result of a simple mistake

Examples:

- Tester failed to carefully compare expected and actual results

Determining Probable Root Causes

Five Whys

- Ask why enough times to get to the root cause of the defect
- Up to 5 repetitions may be needed

Example

- **Consider customer reported feature interaction error**
- **Why was defect not detected during testing**
  - Feature interaction was never tested
- **Why was feature interaction not tested**
  - Late feature addition was not communicated to test team

Developing Possible Solutions

Oversight Problems

- Use of checklists
- Tools to automate checking
- Work-product templates
- Reviews

Developing Possible Solutions (cont'd)

Education Problems

- Just in time training
- Tutorials
- Proper staffing

Developing Possible Solutions (cont'd)

Communication Problems

- Liaisons to other groups
- Change tracking system improvements
- Improved documentation
- Changing processes

Developing Possible Solutions (cont'd)

Transcription Problems

- Tools to automate
- Reviews

Summary

Test Management Part 2

People Management

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Apply best practices for maximizing test team performance

Sources of Leverage for Improving Software Development

Make Effective Use of Your People

- One of the most significant factors of a project's success is the ability, experience and motivation of its people
- Software test leader must do everything possible to maximize individual and team effectiveness

Responsibilities of System Test Lead

- Negotiate with project management on schedule and resources
- Negotiate with development team on entry criteria
- Establish priorities for test team
- Manage team

Basic Management Skills

- Leadership
- Communication
- Delegation
- Negotiation
- Motivation
- Problem solving

Motivation

What factors motivate software testers?

Motivation Guidelines for Leaders

- **Learn what your team's needs, goals and motives are**
- **Provide a work environment and task structure which fulfills people's needs**
- **Implement a fair reward system**
  - Recognition, technical advancement, increased technical knowledge and responsibility are important factors as well as salary
- **If goals, plans, requirements and expectations are clearly communicated and understood by the team, the team members will be motivated to perform**

Key Motivators for Software Testers

Pride in work

- Testers who take pride in their work have increased job satisfaction and higher quality
- Test lead must ensure that testers are “proud” of their work
- Test lead must ensure testers can accomplish their tasks

Key Motivators for Software Testers

Pride in accomplishment

- Early success increases motivation
- Project activities should be structured to create early "accomplishments"

Pride in contribution

- There is a strong human desire to contribute
- Recognition of the importance of each tester's contribution is critical

Pair Testing

- Involves 2 testers exploring system input space for tests to execute
- Analogous to pair programming in extreme programming
- Forces communication and classification of testing strategies
- Advantages and disadvantages

Team Development

It is important to improve the knowledge and skills of all team members

Strategies for team development include:

- Training
- Job rotation
- Mentoring
- Reviews, appraisals and feedback

Importance of Teamwork

A successful project requires skilled individuals who work effectively together as a team

Team success is driven by the clear communication of project goals

- The team must understand what it is trying to accomplish
- Individual team members must understand how they contribute to the team's goal

Characteristics of Good Teams

(derived from Paulish, “Architecture – Centric Software Project Management)

- When a team member is struggling, others automatically help out
- Team interactions are “comfortable”
- Conflict is quickly resolved
- Team member’s know each other well and can anticipate reactions to proposed ideas
- Team members are committed to the project

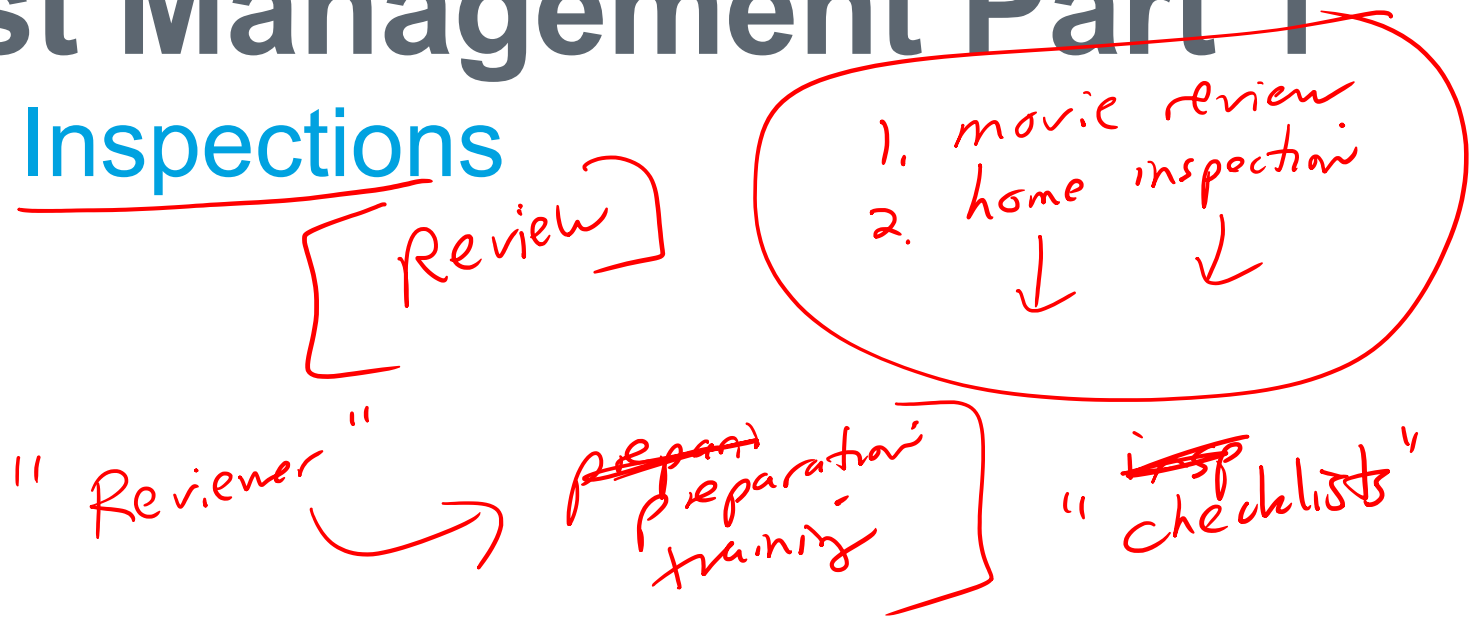
Characteristics of Good Teams (cont'd)

- Team members use "we" rather than "I" in discussing accomplishments and problems
- Team leader is viewed more as a "coach" than a "dictator"
- Team members understand their roles and responsibilities
- Team takes pride in its work
- Team learns from its mistakes
- team believes that it will be successful

Summary

Test Management Part 1

Test Inspections



Objective

Objective

Plan and participate in various work product inspections

test deliverables

System Tester Involvement in Inspections

- System testers should be involved in inspecting requirements documents during development

- System testers should also inspect their work products:
  - Test plan
  - Test case
  - Test Incident Report

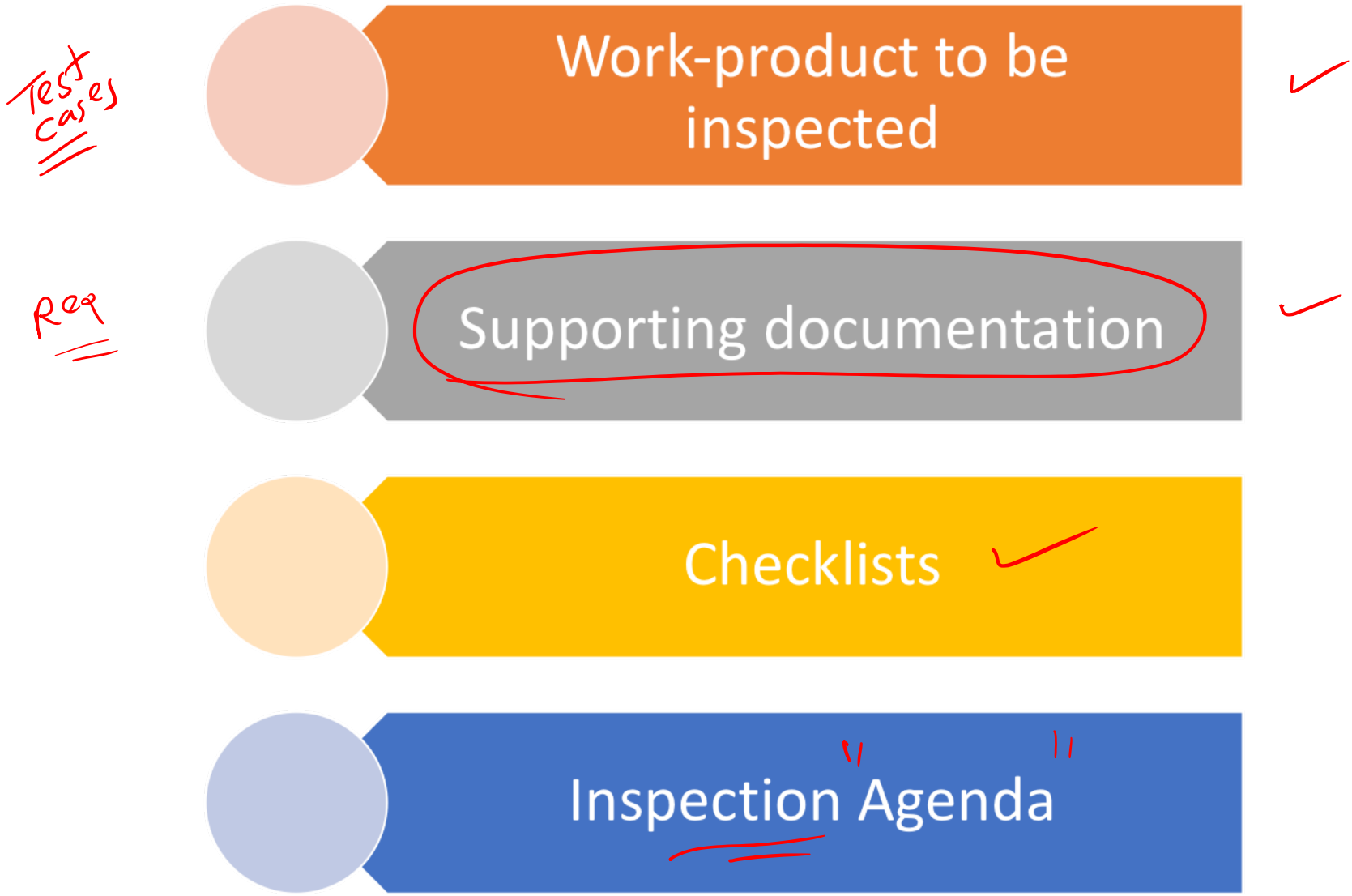
Definitions

An inspection is a "formal statistical process control method for evaluating documents and their production"

Inspections require:

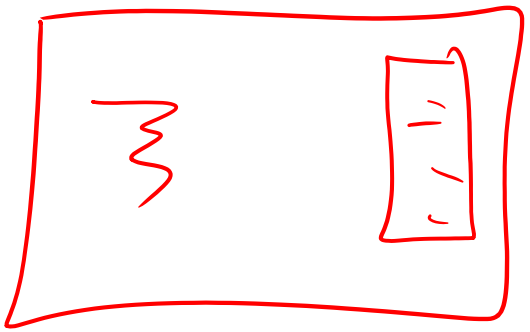
- Advanced preparation
- Utilization of rules and checklists
- Metrics gathering and analysis to facilitate process improvement

Inspection Package



Checklists

- ”Job Aid" for inspectors
- Fundamental tool in the inspection process
- Guides the defect detection process
- Customized by project and document
- Available electronically



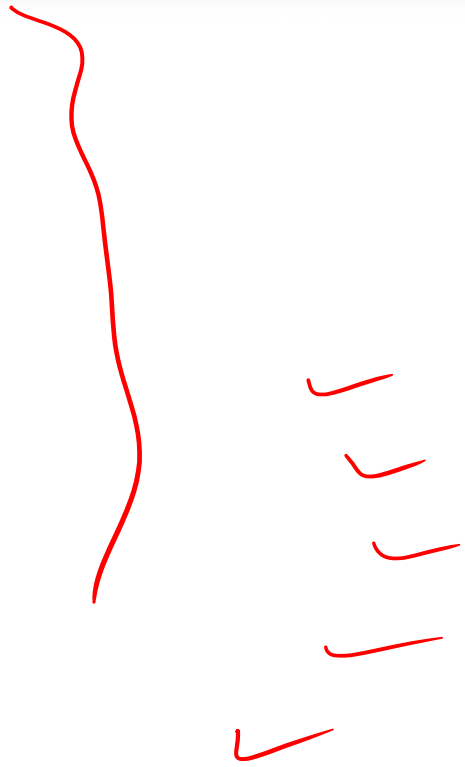
Testability Concerns

- **Testers must advocate for testability features**
- **Testability implies both visibility and control**

- **Examples include:**
  - Diagnostics
  - Test points to view or modify data during execution
  - Ability to place the system into a given state

Requirements Checklist

1. testability
2. Completeness
3. Correct
4. consistent
5. clarity
6.
7.



Test Plan Checklist

1. test objectives defined / clear
2. test strategy
3. reasonable schedule
4. test environment defined
5. test priorities identified
6. ↓ ↓
7.

Test Case Checklist

Bugs documentation

well defined

1. repeatable
2. defined inputs / expected results
3. environment / state
4. sequencing
5.
6.
7.

Summary

Test Management Part 2

Test Maturity Model

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Assess the maturity level of a testing organization.

Applicable Maturity Models

- CMMI
- TMMI

Capability maturity model

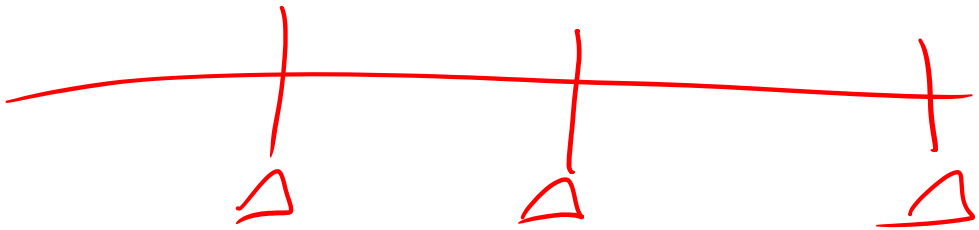
cmu - SEI

1 - 5

self assessment

CMMI Levels

- Initial: adhoc
- Repeatable: expertise lies in individuals
- Defined: processes are defined and documented
- Managed: metrics are used extensively to guide process
- Optimizing: emphasis on continuous improvement



TMMI

Test maturity model Integrated

- Developed in an analogous manner to assess maturity of testing process
- Five levels of maturity defined
- Each level has goals and subgoals that are achieved via ATRs (activities, tasks, responsibilities)

1 - 5

TMMI Level 1: Initial

- Ad hoc testing without specific goals
- Test process begins after code is written
- Testing is not a priority
- Test tracking is not performed

TMMI Level 2: Phase Definition

- Software testing plans are developed that include well defined testing phases
- Basic testing methods are introduced
- Goals for testing are defined

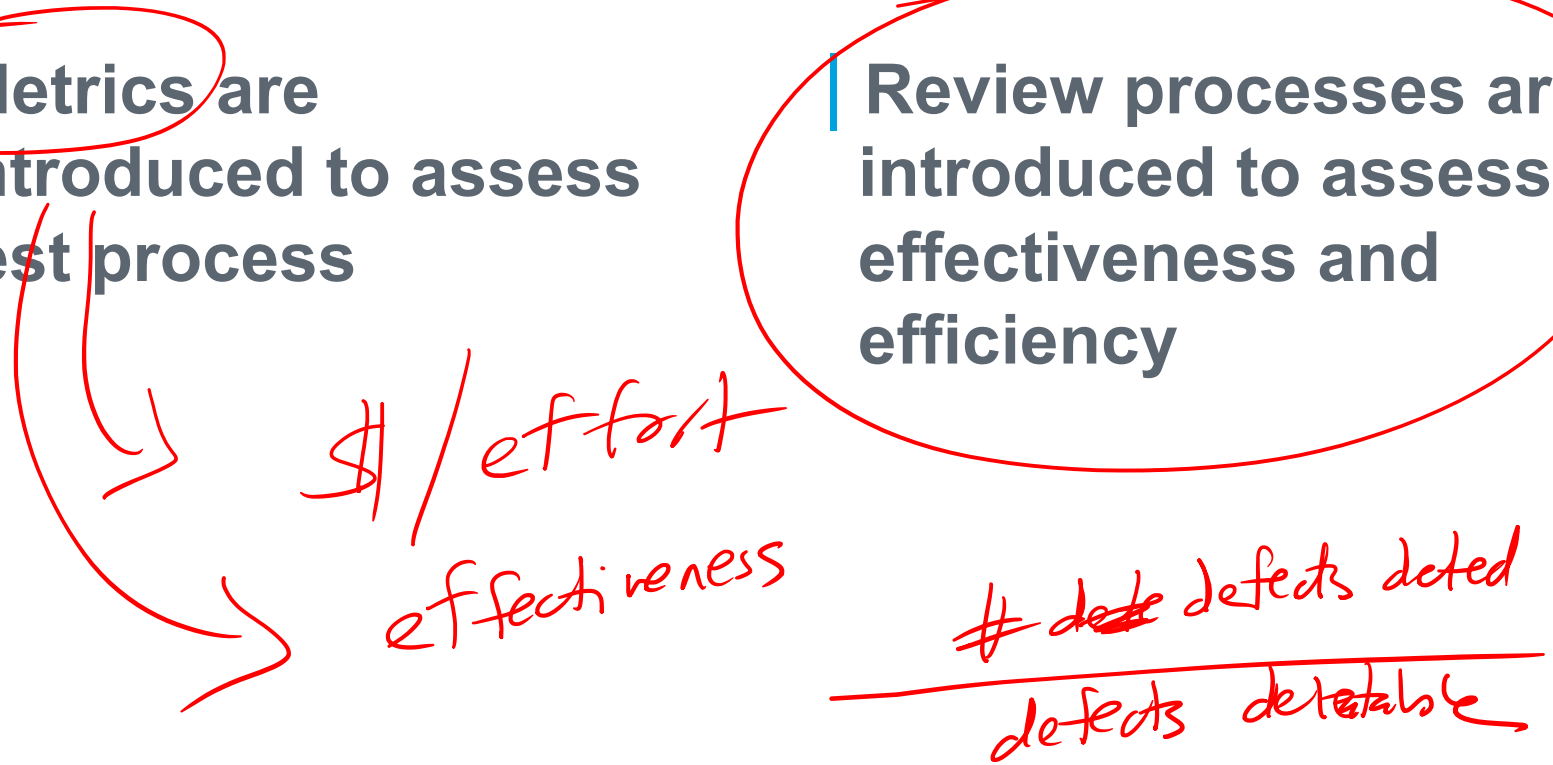
TMMI Level 3: Integration

- Software testing is integrated into the software development process
- Test progress is monitored and tracked
- Testers are trained
- Test plan risk management is performed

prioritize

TMMI Level 4: Management and Measurement

- Metrics are introduced to assess test process
- Review processes are introduced to assess effectiveness and efficiency



TMMI Level 5: Optimization, Defect Prevention and Quality Control

- Root cause analysis is performed to prevent defects from reaching the customer
- Statistical quality control used to monitor test process
- Test process improvement implemented

Summary

CMMI ≈ TMMI

Test Management Part 2

Test Outsourcing

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Develop a plan for outsourcing testing.

Home Analogy

- Do you do the work yourself or hire a contractor to do the work for you?

Factors Influencing Outsourcing Decision

- Need for specialized / advanced technology or capability
- Strategic value of system
- Cost
- Strategic Alliance
- Maintenance support
- Speed of development
- Desire for level staffing

Test Outsourcing Activities

- Define the work to be subcontracted
- Develop an outsourcing plan
- Perform domain and/or process evaluations if needed
- Select a subcontractor
- Contract with subcontractor
- Oversee the testing to ensure that it is on schedule, within cost, and meeting requirements
- Accept the testing

Defining Subcontracted Work

Partition the product and its activities

Select subcontracted work to:

- Maximize organization's effectiveness
- Match skills and capabilities of potential subcontractors
- Minimize communication and coordination efforts
- Minimize dependencies
- Minimize risk of subcontractor knowledge not being transferred to the organization

Develop a Subcontractor Management Plan

Develop a technical specification for the work

Develop a statement of work for the supplier

- Identify all tasks to be performed
- Identify relevant processes to be followed
- Identify maintenance responsibilities

Develop a Subcontractor Management Plan (cont'd)

Perform risk management

- Identify outsourcing risks early
- Adjust acquisition strategy to minimize risks

Estimate resources needed

- Estimate supplier test effort (what it should cost)
- Estimate effort for vendor

Components of an Outsourcing Agreement

- Contract
- Statement of work
- Technical specs

Supplier Management Activities

Selection

- Searching for and evaluating vendors
- Specifying legal terms of contract
- Negotiating contract

Subcontract management

- Conveying and explaining requirements
- Monitoring vendor, including reviews
- Resolving problems

Factors to Consider in Selecting a Supplier

- Strategic business alliances
- Prior performance on similar work
- Geographic location
- Software testing and software management capabilities
- Available staff
- Domain expertise
- Cost
- Similarity of processes and tools
- Business viability

Contract with Supplier

- See lawyers!

Tracking and Oversight

A plan must be developed for supplier tracking and oversight

Plan must address:

- Activities to be performed and the schedule to perform them
- Identification of groups, assigned responsibilities and inter-group communication
- Techniques, tools and methodologies to be employed for review and tracking of vendor performance
- Escalation procedures

Tracking and Oversight Activities

- Maintaining good communication
- Reviews
- Metrics
- Continuous risk management
- Approval of invoices

Acceptance of Work

- Acceptance of work is a formal procedure which has contract implications
- You must be convinced that the testing is thorough and complete

Summary

Test Management Part 2

Test Process Improvement

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

→ faster
→ effectively

Objective

Explore Strategies for Improving the Testing Process

Improving the Testing Process

Software test teams must constantly look for ways of testing faster and more effectively

"ASU Football"

Process improvement phases

- Characterize the current process
- Analyze current process
- Characterize target process
- Process redesign
- Implement

Characterizing the Current Process

It is important to understand the current testing process

Care must be exercised in distinguishing among:

- The Perceived Process: What you think you do.
- The Official Process: What you are supposed to do.
- The Actual Process: What you do

Quote from Watts Humphrey

SEI "CMM"

"If you don't know where you are, a map won't help."

Analyze Current Process

- Identify value-added and non-value added activities
- Identify the sources of variation in the process
- Software metrics play a key role in objective process analysis
- Software test leader must plan to collect the appropriate metrics

Testing Metrics

You cannot control what you cannot measure

GQM paradigm (goal-question-metric)

- Define the goals of the measurement process
- Derive the questions that must be answered to meet the goals
- Develop metrics to answer the questions

Example Test Goal

- Reduce testing time
- Find more severe defects

"Reduce APS utility bill during the summer"

- How much energy? kWH
- When is it used?
- Where?

Example Testing Questions

- Where do we spend most of our time during testing?
- What types of defects are we missing?
- What is our testing productivity

Possible Testing Measures

- Test productivity
- Test quality

Characterize Target Process

Identify the ultimate goal of the process e.g.,

- Minimize test time
- Find more high severity problems

Quote from Watts Humphrey

- "If you don't know where you are going, any road will do."

SQM

Process Redesign

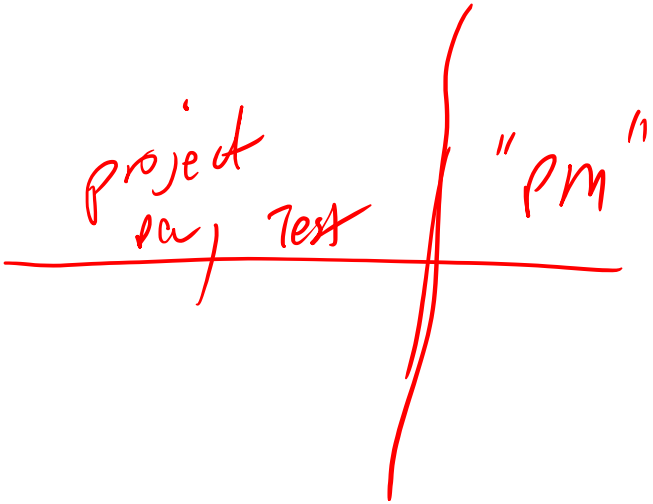
- Improve the current process in the direction of the target process
- Explore ways for eliminating, simplifying or combining activities
- Explore ways of eliminating rework
- Explore ways of reducing task variance

Implement Process Improvements

- **Set process improvement goals**
- **Implement process improvements**
  - Begin with candidates that are well defined
  - Pilot candidates that are not proven
- **Measure progress towards goals**

Process Improvement Tools - Post Mortem / Lessons Learned / Retrospectives

A set of lessons learned is documented



Analysis techniques include:

- Interviews with key personnel
- Statistical analysis of data
- Investigations of major problems
- Identification of what went well and what didn't go well

Testing Lessons Learned Addresses

- Overall schedule analysis
- Adequacy of entrance criteria
- Adequacy of exit criteria
- Overall quality
- Team interactions

Example Questions to Address

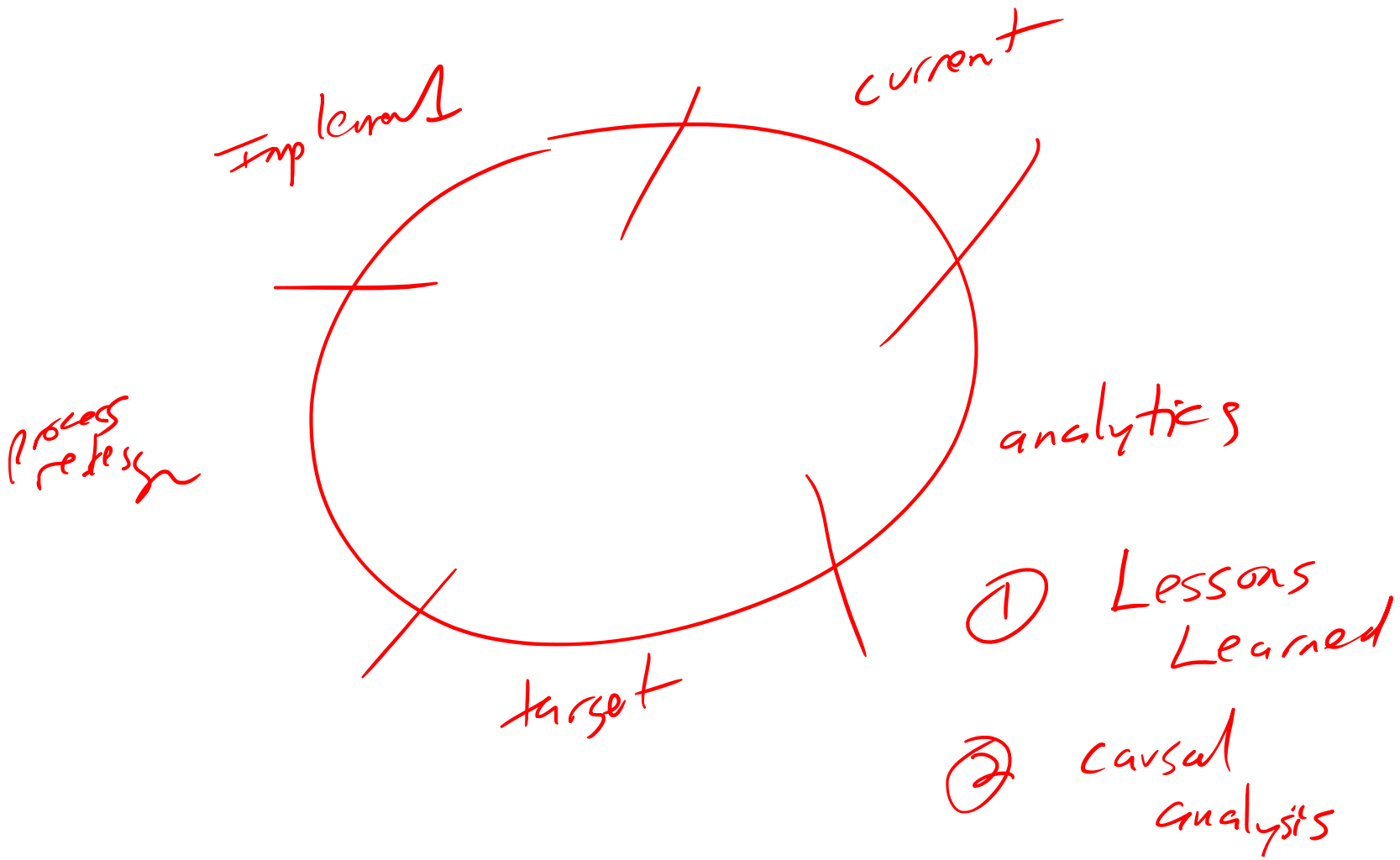
- How effective was communication and collaboration with development?
- How effective was test estimation?
- Are there any issues with the test environment?
- What types of defects are we missing?
- Where are we wasting time?

Process Improvement Tools - Causal Analysis

- Seeks to identify the root cause of defects as well as approaches to eliminate future occurrences

- Covered in another module

Summary


current
analytics
target
process redesign
① Lessons Learned
② causal analysis

Test Management Part 2

Test Tracking

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Utilize various measures to track testing progress.

System Test Tracking

- **During system test information must be collected and tracked to assess:**
  - Product quality (previously discussed under "system test exit criteria")
  - Testing progress
- **Numerous measures exist for tracking test progress against a plan including:**
  - Percentage of tests developed
  - Percentage of tests executed
  - Percentage of requirements tested
- **Testing schedule and effort progress can also be assessed via earned values**

Earned Values

- **Earned values are a technique for tracking both schedule and cost progress**
- **Earned value approach establishes a relative value for every task and credits that value when the task is completed**
- **Progress is then tracked in terms of:**
  - BCWS (Budgeted Cost of Work Scheduled)
  - BCWP (Budgeted Cost of Work Performed)
  - ACWP (Actual Cost of Work Performed)

Earned Value Example

Assume that we are concerned with testing two concurrently developed increments of a product labeled "1" and "2" and for each increment there are features labeled "A" - "F". Thus, task "1A" corresponds to the activity of testing feature "A" in increment "1". Associated with each testing task is an earned value corresponding to the estimated effort to complete the task.

Tasks	EV	Tasks	EV
1A	50	2A	30
1B	40	2B	40
1C	30	2C	30
1D	20	2D	50
1E	50	2E	40
1F	30		

Earned Values Example
(continued)

Assume the following schedule:

Week 1	Week 2	Week 3	Week 4
1A, 1B	1C, 1D	1E, 1F	
	2A, 2B	2C, 2D	2E

Earned Values Example
(continued)

Week	Work Completed	Cost
1	1A, 1B, 1C	100
2	1D, 2A, 1E	70

Earned Values Example
(continued)

	Week 1	Week 2	Week 3	Week 4
BCWS	90	210	370	410
BCWP	120	220		
ACWP	100	170		

Earned Values Example
(continued)

- **Based on earned values the project is ahead of schedule at the end of week 2**

BCWS	=	210
BCWP	=	220

- **Based on earned values the project is below budget at the end of week 2**

BCWP	=	220
ACWP	=	170

Summary