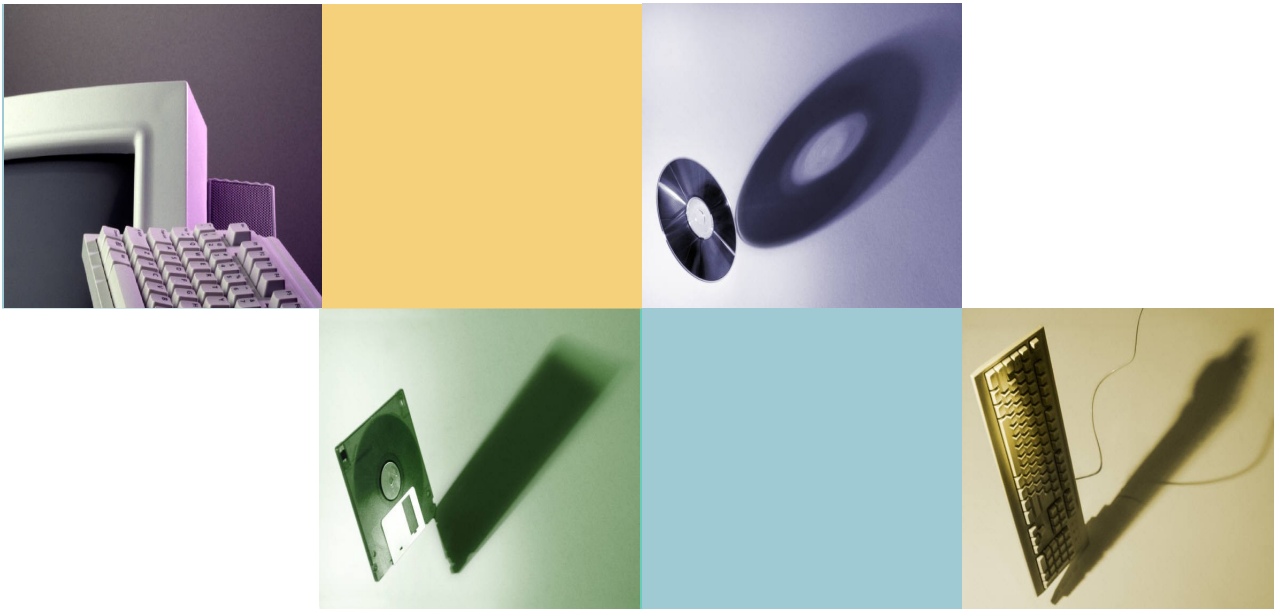


The Basics of Structured Software Testing



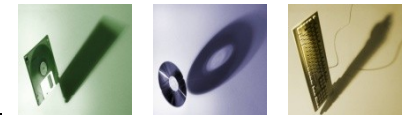
Craig Mayer
Director, Sogeti USA
Spring 2007

Basics of Structured Software Testing

- Goals and objectives
- What is testing?
- Test levels and test infrastructure
- Structured software testing
- Benefits of structured software testing



- Goal
 - Provide a background to risk-based, quality-centric testing
- Objectives
 - Introduce the basic concepts behind and importance of risk-based, quality-centric testing (“structured software testing”), particularly as it applies to System Integration Testing (SIT)
 - Explain why a focus on quality throughout a software development project, from requirements to post-production, results in better software for users
 - Provide an overview of the lifecycle of structured software testing and key artifacts created in each phase of testing
 - Explain the benefits of structured software testing



What is testing?

Testing is a process aimed at:

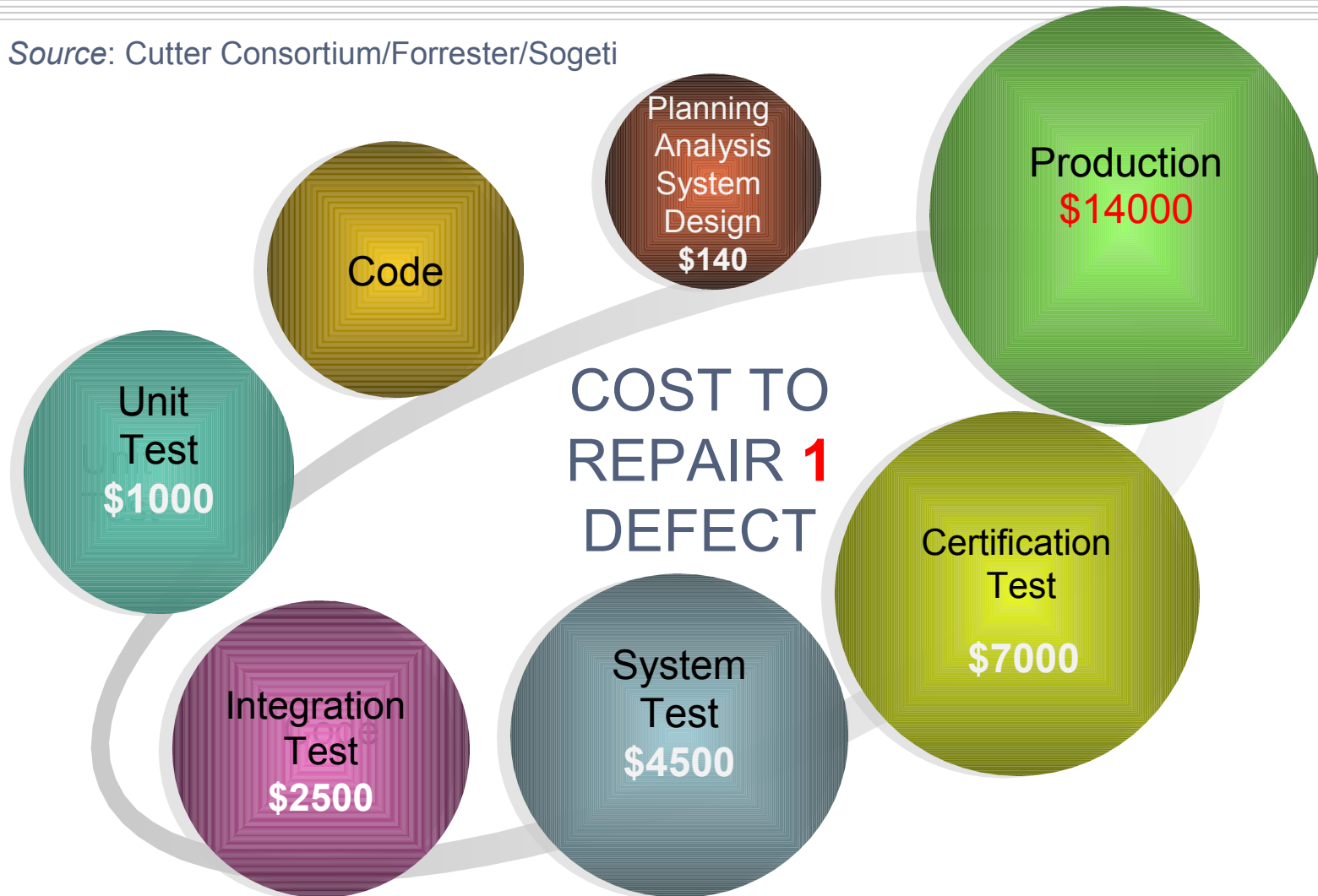
- Finding defects *in a controlled manner*
- Detecting the level of quality of the test object
- Demonstrating the gap between specifications and the actual product
- Demonstrating that the end product functions as called for in the requirements



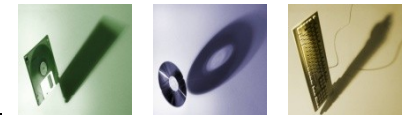
How Quality Provides Tremendous Savings



Source: Cutter Consortium/Forrester/Sogeti



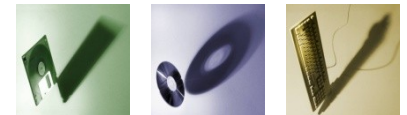
Time cost and resource effort increase exponentially later in the lifecycle



Testing is not

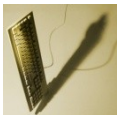
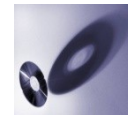
Testing is not:

- Implementation
- Acceptance
- Defect repair
- A phase *after* development – although testing includes a phase after development
- Training on a new system



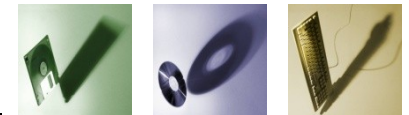
Why do we test ?

- To mitigate business risks that have been identified:
 - Validate software quality
 - Verify integration of business processes
 - Reduce time-to-market
 - Competitive purposes
- To ensure business usability of the software:
 - Release low-error/known-error software
 - Move high quality software (meets or exceeds quality expectations) into production
 - Demonstrate the usefulness of new technology



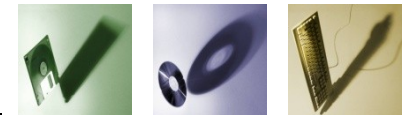
What do we test?

- Application Software
- Hardware
- System Software
- Procedures
- Documentation
- Functionality
- Continuity
- Performance
- Usability
- Interoperability (between different applications and systems)

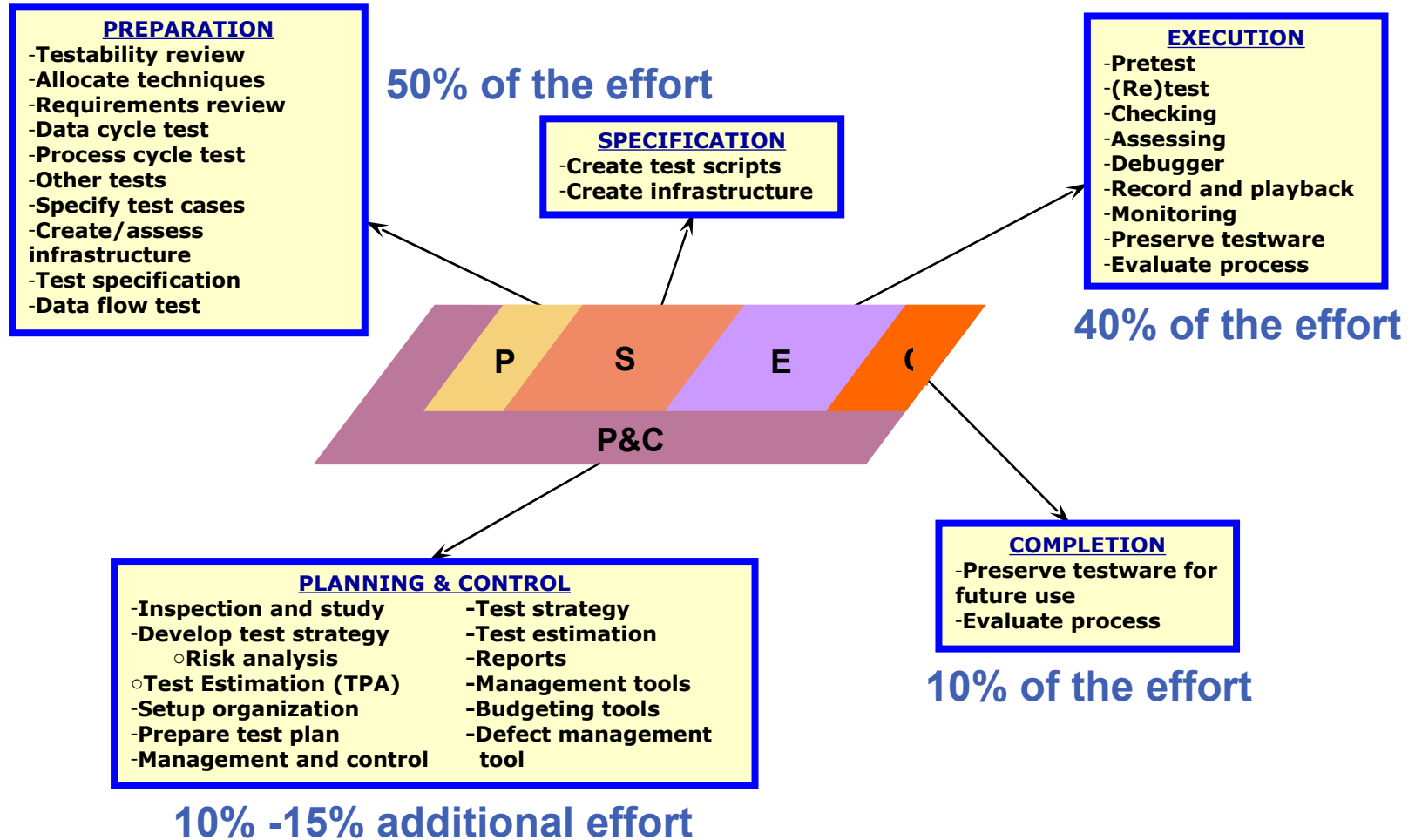


Ambiguous/poor requirements

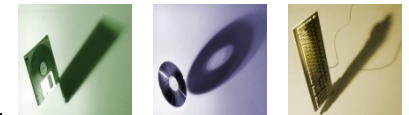
- Ambiguous and improperly written (and, subsequently, poorly coded) requirements cause major headaches for a project when encountered, especially during SIT
- SIT only tests for required functionality – nothing more, nothing less
 - Thus inadequate requirements will be tested “as is”; quality expectations may not be met!
- SIT is based on the requirements as documented in different classes of requirements and design documentation



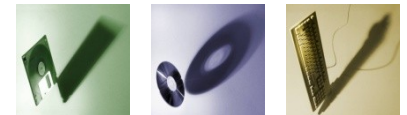
TMap® - Key artifacts by testing phase of the structured testing lifecycle



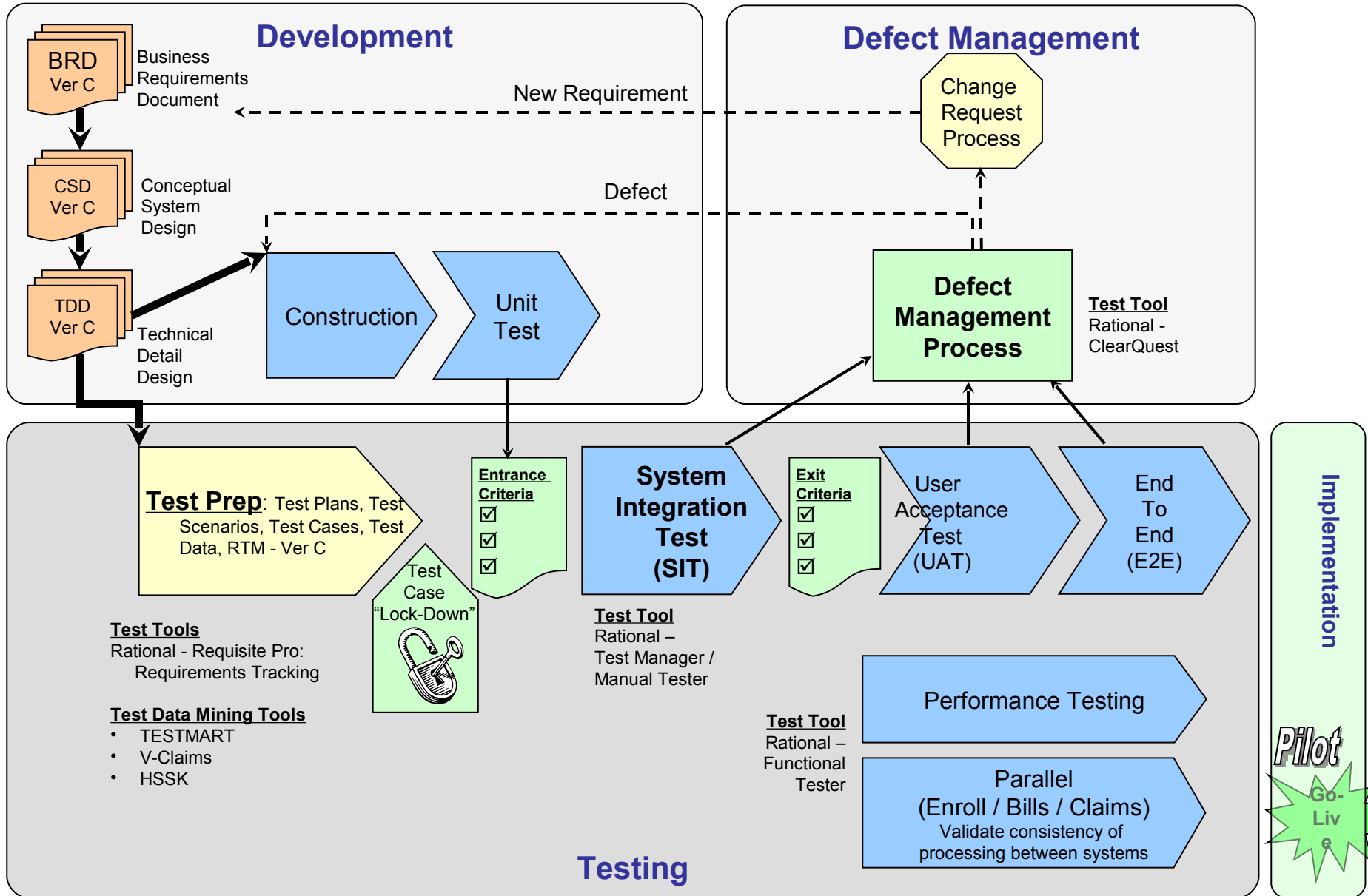
- Goal: demonstrate that the developed system meets the business requirements
- Document classes typically comprising the SIT test basis:
 - Business Requirements Documents
 - Conceptual System Design Documents
 - Technical Detailed Design Documents
- Resulting assets:
 - A test plan
 - Test scenarios
 - Test cases
 - Test scripts
 - Requirements traceability matrix (RTM)
 - Defects log and defect metrics



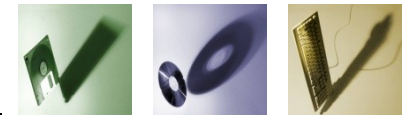
- A SIT plan is typically created to test the integration of the functional areas comprising a broader system and the quality of the broader system taken as a whole
- A SIT plan includes:
 - Specific goals, objectives, and approach, including identification of subsystems
 - Facilitated sessions: to identify/prioritize risks and relevant quality characteristics
 - Depth of SIT testing to validate quality for each subsystem is dependent on the risks and quality characteristics associated with the functional area/subsystem
 - If needed, a “generic” SIT plan (common to all functional areas) should also be created + a SIT plan for each functional area



Integration: Development, Testing & Defect Management



- Testing everything is impractical
 - Funding limitations
 - Limited time and resources
 - Diminishing value after a certain point
 - Isn't there a more effective way?
- Yes! Structured software testing: a risk-based, quality-centric approach to testing



To be effective, structured software testing should be:

- Risk-based

- Focuses on testing most thoroughly, and thereby mitigating, the highest risk elements of the product and project

- Quality-centric

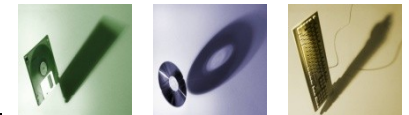
- Focuses on creating a quality product at each step of the software development lifecycle

- Seeks defects in documentation, coding, testing, environments, and deployment as early as possible, logging them for review/remediation by the business and technical teams

- Pursues validation of quality throughout all phases of the development lifecycle



- SIT should be based on the mitigation of risk and validation of expected quality defined by the business requirements
- Definition of risk
 - A “risk” is a chance of a failure occurring, related to expected damage to the project (should the failure occur)
- Risk ranking
 - Risks identified in different categories (e.g., business risks, project risks, product risks, process risks) may be ranked in criticality relative to one another, by instituting a method of risk ranking (or “risk rating”)



How risk ranking works

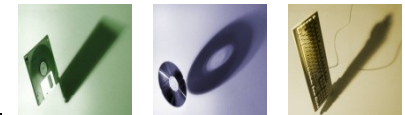
- Assemble a team of people representing various roles from the project (PMs, Business Leads/SMEs, Test Leads, [Tech Leads])
 - A. Create an initial list of risks
 - B. Team assigns a numeric value to each risk (scale of 1-5 or 1-10) = the probability of occurrence of each risk
 - C. Team assigns a second value to each risk, representing the impact on the project/organization should that risk occur
 - D. Multiply the two values together (the probability of occurrence X the impact)
 - The result is a relative value for each risk (at the time it was identified) compared with any other risk
 - E. Order the risks by their relative risk values (“risk rating” or “risk ranking”)
 - Ranking helps manage the most critical risks, especially those falling in the middle tier of the ranking (i.e., those with either a low probability of occurrence/high impact should they occur or those with a low impact should they occur/high probability of occurrence)
- Periodically update the list of risks



Risks - quality characteristics matrix

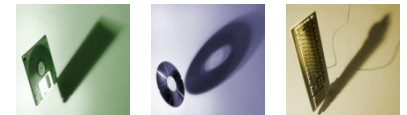
Multi-Level Matrix

Quality Characteristics	High Priority	High Priority/ Low Risk	High Priority/ Medium Risk	High Priority/ High Risk
	Medium Priority	Medium Priority/ Low Risk	Medium Priority/ Medium Risk	Medium Priority/ High Risk
	Low Priority	Low Priority/ Low Risk	Low Priority/ Medium Risk	Low Priority/ High Risk
		Low	Medium	High
		Risks		



Quality characteristics and their relative importance

- Identified risks, detailed requirements, and acceptance criteria are collectively referred to as “the why” of SIT
 - They form the basis for formulating test goals
- A “test goal” is defined as:
 - A goal that is relevant to the organization (a reason for testing)
 - A goal that is formulated in terms of:
 - Business processes supported by IT
 - Realized by user requirements or use cases, critical success factors, change proposals, or cited risks for coverage by the test effort
 - Concentrating test effort on validating the presence of prioritized quality characteristics is a key goal for a SIT plan



Allocating quality characteristics: how to do it

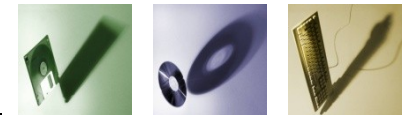
- Determine which quality characteristics are relevant and applicable to the system under test
- Determine the relative importance of each quality characteristic
- Divide each functional area into subsystems and determine their relative importance
- Create subsystem/quality characteristic pairs
- Specify the relative test importance (test depth) per subsystem/quality characteristic
- Establish the test technique to be used for each pair



Industry standard quality characteristics

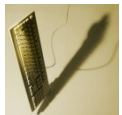
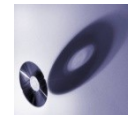


	TMap Quality Characteristics	ISO 9126
1	Connectivity	Interoperability
2	Continuity	Reliability
	Degradation Possibilities	Fault Tolerance
	Possibility of Diversion	Recoverability
	Operation Reliability	Maturity
	Recoverability	Recoverability
	Robustness	Fault Tolerance
3	Data Controllability	Suitability
4	Effectivity	Learnability
		Operability
		Suitability
		Understandability
5	Efficiency	Resource Utilization
6	Flexibility	Suitability
7	Functionality	Accuracy
8	(Suitability of) Infrastructure	--
9	Maintainability	Analyzability
		Changeability
10	Manageability	Operability
11	Performance	Time Behavior
12	Portability	Adaptability
		Installability
13	Reusability	--
14	Security	Security
15	Suitability	Suitability
16	Testability	Testability
17	User-Friendliness	Attractiveness
		Learnability
		Operability
		Understandability

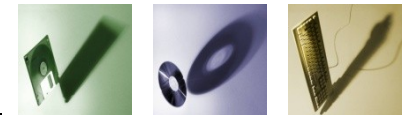
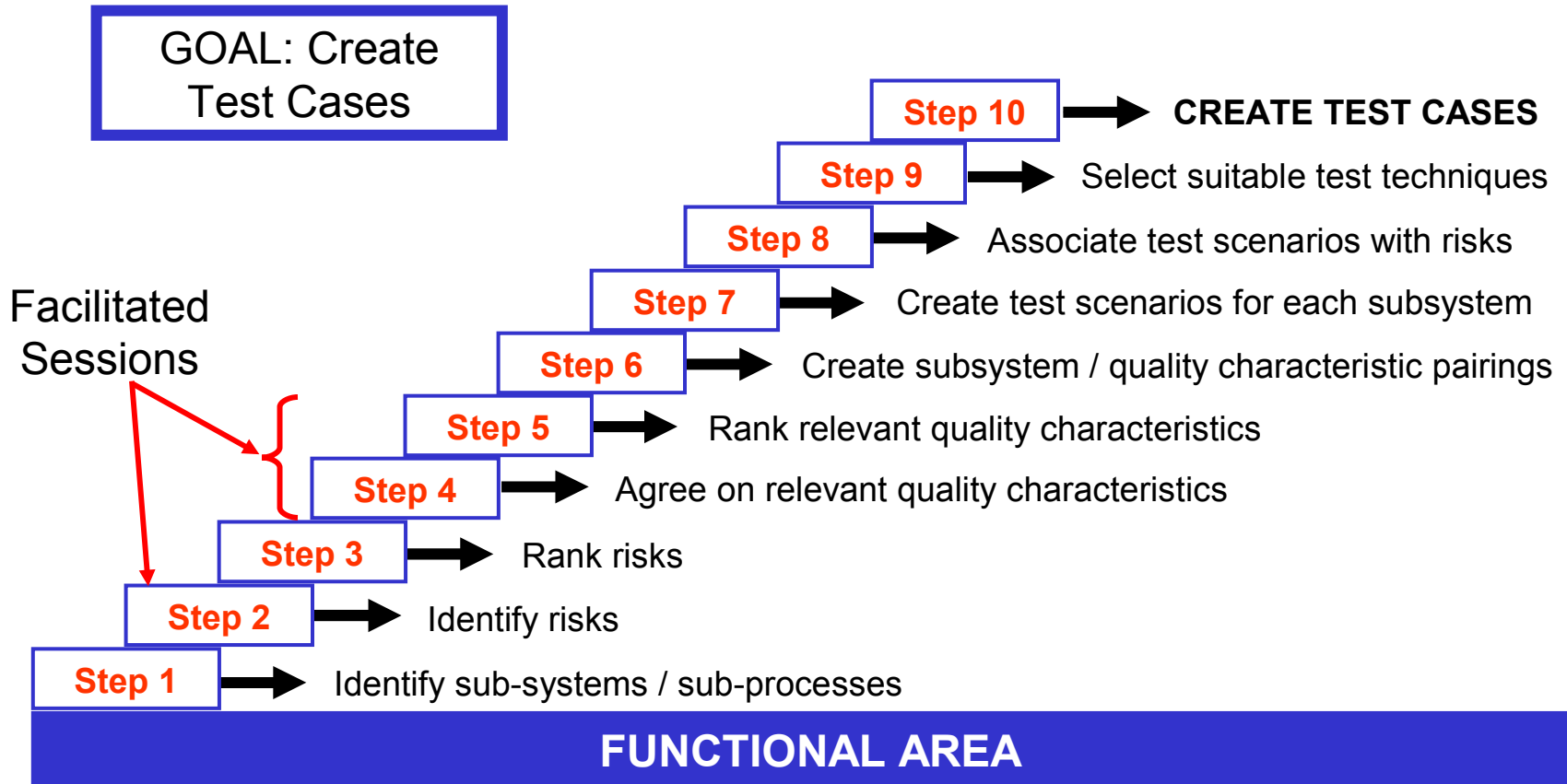


Quality characteristics - definitions

- **CONNECTIVITY**
The ease with which a link with a different information system or within the information system can be made and modified.
- **CONTINUITY**
The certainty that data processing will continue uninterruptedly, which means that it can be resumed within a reasonable period of time even after serious interruptions.
- **DATA CONTROLLABILITY**
The ease with which the correctness and completeness of the information (in the course of time) can be checked.
- **EFFECTIVITY**
The degree to which the information system meets the demands of the organization and the profile of the end users for whom it is intended, as well as the degree to which the information system contributes to the achievement of business objectives.
- **EFFICIENCY**
The relationship between the performance level of the system (expressed in the transaction volume and overall speed) and the amount of resources (CPU cycles, I/O time, memory and network capacity, etc.) that are used.
- **FLEXIBILITY**
The degree to which the user may introduce extensions or modifications to the information system without changing the software itself.
- **FUNCTIONALITY**
The certainty that data processing is correct and complete, in accordance with the description in the functional specifications.
- **(SUITABILITY OF) INFRASTRUCTURE**
The suitability of hardware, network, systems software and DBMS for the application concerned and the degree to which the elements of this infrastructure interrelate.
- **MAINTAINABILITY**
The ease of adapting the information system to new demands from the user, to changing external environments, or in order to correct defects.
- **MANAGEABILITY**
The ease with which to get and keep the information system in its operational state.
- **PERFORMANCE**
The speed with which the information system processes interactive and batch transactions.
- **PORTABILITY**
The diversity of the hardware and software platforms on which the information system can run, and how easy it is to transfer the system from one environment to another.
- **REUSABILITY**
The degree to which parts of the information system, or the design, can be reused for the development of different applications.
- **SECURITY**
The certainty that data can be viewed and changed only by those who are authorized to do so.
- **SUITABILITY**
The degree to which manual procedures match the automated information system and the fitness for use of these manual procedures for the organization.
- **TESTABILITY**
The ease with which the functionality and performance level of the system (after each modification) can be tested and how fast this can be done.
- **USER-FRIENDLINESS**
The ease with which end-users use the system.



Ten steps for creating a test case

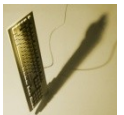
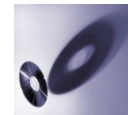


Test Techniques



TMap technique	•White-box / Black-box (WB / BB), formal / informal	•Test basis	•Derivation principle (with coverage): Processing logic, Equivalence partitioning, Operational usage, CRUD, Other	•Quality characteristics	•Application areas
•Algorithm test	•WB, formal	•Internal structure, for example, program code or technical design	•Processing logic: decision coverage in combination with path coverage, optionally with additional test measure 2 or higher	•Functionality	•Processing
•Data cycle test	•BB, informal	•Functional requirements	•CRUD: on the basis of the life cycle of data	•Functionality	•Integration between functions and data
•Data combination test	•BB, informal	•Functional requirements	•Equivalence partitioning	•Functionality •Data controllability	•Processing, integration between functions and data
•Decision table test	•WB and BB, formal	•Decision tables, both internal structure and (functional) specifications	•Processing logic: decision coverage in combination with path coverage, optionally with additional decision/condition coverage and/or a higher test measure	•Functionality	•Complex processing
•Elementary comparison test	•WB and BB, formal	•Internal structure (WB) or formal functional requirements (BB), for example, pseudocode or structured language	•Processing logic: Modified decision/condition coverage	•Functionality •Data controllability	•Complex processing
•Error guessing	•BB, informal	•All types of test basis	•Other: based on assumptions as to where the errors have been made	•Security •Data controllability •Functionality •User-friendliness •Suitability •Performance •Efficiency	•All
•Program interface test	•WB, formal	•Internal structure, for example, program code or technical design	•Equivalence partitioning	•Functionality	•Integration between programs
•Process cycle test	•BB, formal	•Administrative Organizational procedures	•Processing logic: decision coverage, standard with test measure 2, but as desired the weight of this test measure can be increased or decreased	•Security •Effectivity •User-friendliness •Suitability	•Integration between the administrative organization and the system
•Real-life test	•BB, informal	•All types of test basis	•Operational usage	•Security •Continuity •Infrastructure •Performance •Efficiency	•Simulation of practical use
•Semantic test	•BB, formal	•Functional requirements	•Equivalence partitioning on the basis of the relationships between data and input	•Security •Functionality •User-friendliness	•Interaction (screens) between system and user, input validation, easy processing
•Syntactic test	•BB, formal	•Functional requirements	•Testing the layout of the screens and reports, and the primary data definitions (this latter is executed on the basis of equivalence partitioning)	•Functionality •User-friendliness	•Interaction (screens, reports, on-line) between system and user, input validation

- **Defects are found early**, costing less time/money to reach production → delivering a higher quality product
- Required **quality of the various test objects is tested for and validated**, by focusing on testing for quality as a risk mitigation strategy
- By **keeping a larger percentage of the testing process/effort off the critical path**, faster time-to-market results
- Structured software testing is **more cost-effective and efficient** than non-structured testing approaches
- **Sound test coverage is provided**, without the need to overlap phases
- **Establishes a test organization** that is prepared and efficient
- Delivers a **repeatable process**



The Basics of Structured Software Testing

Thank you!

