IT3205: Fundamentals of Software Engineering (Compulsory)

BIT – 2nd Year

Semester 3

IT3205: Fundamentals of Software Engineering

Software Testing and Quality Assurance

Duration: 8 hours

Learning Objectives

- State the software testing process and required documentation
- Explain the different software testing techniques and integration strategies
- Design test cases and write test programs for a given simple software problem
- Describe the code verification techniques
- Describe the importance of software quality
- Distinguish the difference between product quality and process quality
- Describe some important quality standards with respect to software

Detailed Syllabus

6.1 Verification and Validation

6.2 Techniques of testing

6.2.1 Black-box and White-box testing

6.2.2 Inspections

6.3 Levels of testing

6.3.1 Unit testing

6.3.2 Integration Testing

6.3.3 Interface testing

6.3.4 System testing

6.3.5 Alpha and beta testing

Detailed Syllabus

- 6.3 Levels of testing
  - 6.3.6 Regression testing
  - 6.3.7 Back-to-back testing and Thread testing
  - 6.3.8 Statistical Software Testing
  - 6.3.9 Object Oriented Testing
- 6.4 Design of test cases
- 6.5 Quality management activities
- 6.6 Product and process quality
- 6.7 Standards
  - 6.7.1 ISO9000
  - 6.7.2 Capability Maturity Model (CMM)

6.1 VERIFICATION AND VALIDATION

Validation and Verification

- Validation and Verification (V & V) is the name given to the checking and analysis processes that ensure that software conforms to its specification and meets the needs of the customers who are paying for the software.
- V & V is a whole life-cycle process. It starts with requirements reviews and continues through design reviews and code inspections to product testing. There should be V& V activities at each stage of software process.

- **Validation:** Are we building the right product?
- **Verification:** Are we building the product right?

Validation and Verification

- Within the V & V process, two techniques of system checking and analysis may be used:

1. **Software Inspections**
   - Analyze and check system representations such as the requirements documents, design diagrams and program source code. They may be applied at all stages of the development process. Inspections may be supplemented by some automated analysis of the source text of a system or associated documents. Software inspections and automated analysis are static V & V techniques as they do not require the system to be executed.
2. **Software Testing**
   - Involves executing an implementation of the software with test data and examining the outputs of the software and its operational behavior to check that it is performing as required. Testing is a dynamic technique of V & V because it works with an executable representation of the system.

Software Testing Procedure

- Testing procedures should be established at the start of any software project. All testing carried out should be based on a test plan, this should detail which tests are to be carried out.
- For each test, the following information should be included in the test plan:
  - the pre-requisites for the test.
  - the steps required to carry out the test.
  - the expected results of the test.

- The outcome of any tests should be recorded in a test results document that include whether the test succeeded or failed and a description of the failure. Test results for all passes through the test plan must be recorded to allow accurate records to be kept of where problems occur and when they were identified and corrected.

Testing Process

1. Run the tests as defined by the test plan.

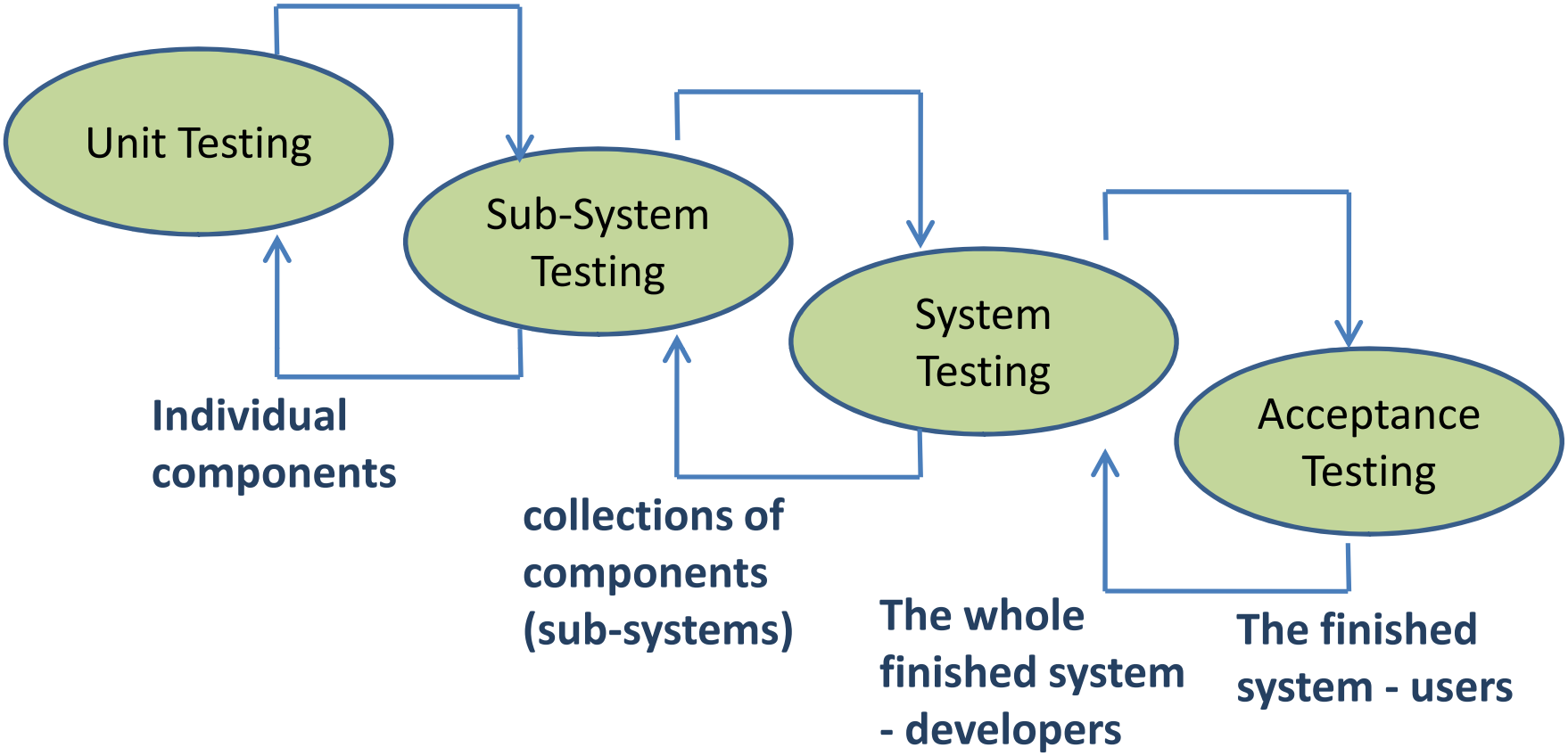
   Note: Testing should not stop when the first problem is encountered, unless it is so severe that the rest of the tests would be meaningless. Rather all testing in the test plan should be carried out and then the errors addressed.

2. Record the outcome of each test in the test report, both success and failure should be reported. For failed tests the nature of the problem should be described in sufficient detail to allow it to be corrected and to allow analysis of the types of errors being found.
3. Correct the errors that were documented from the test run.
4. Repeat the process until no errors are identified or error rate is sufficiently low. If the error rate is low then it may be sufficient to simply re-test the failed errors. If the error rate is high then all tests should be re-run.

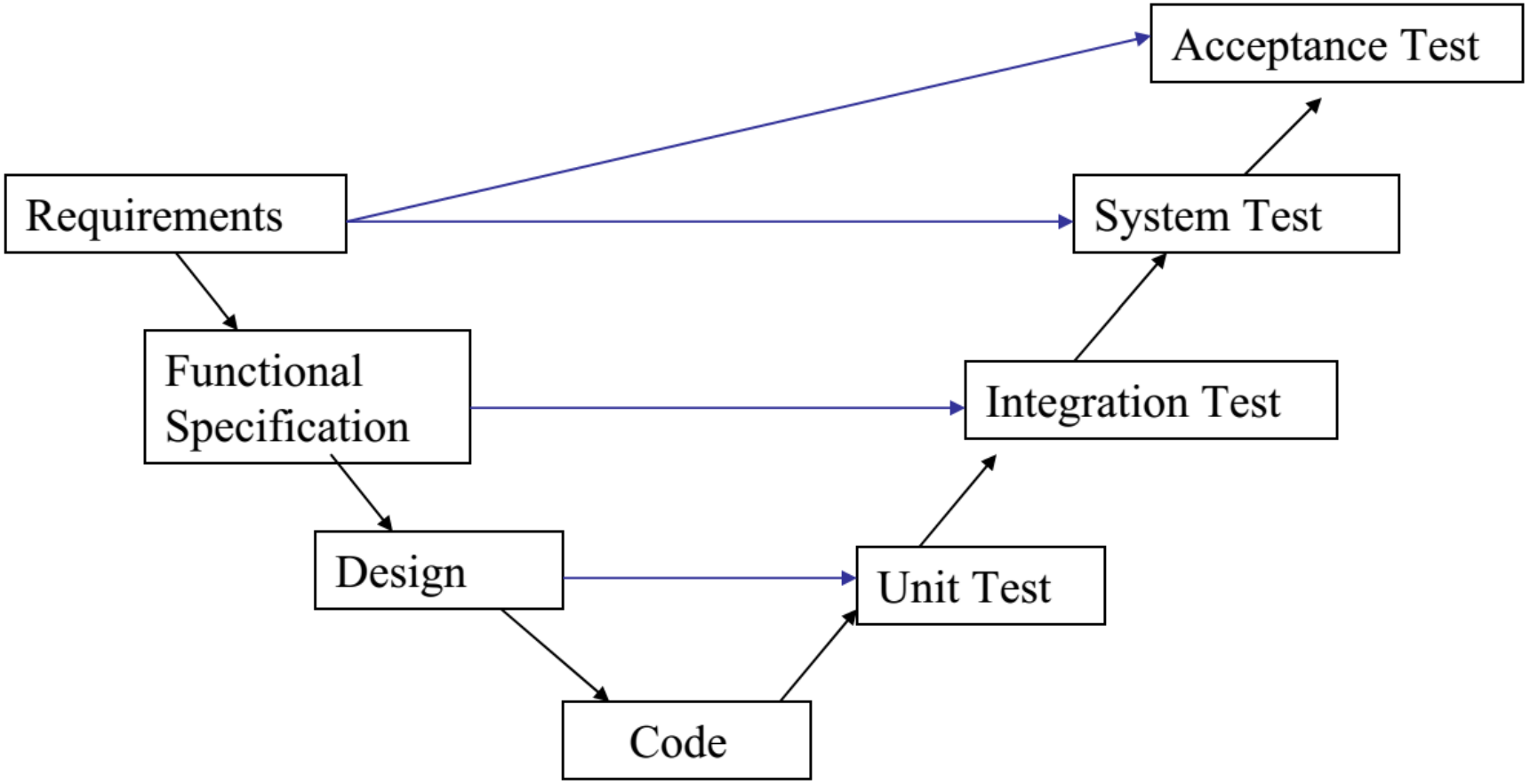
Dynamic & Static Verification

- Dynamic verification
  - Concerned with exercising and observing product behavior (testing)
    - includes executing the code
- Static verification
  - Concerned with analysis of the static system representation to discover problems
    - does not include execution

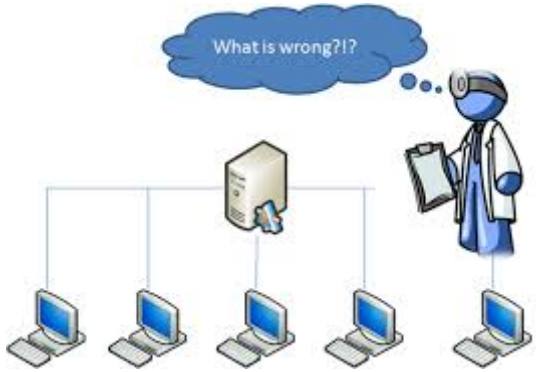
Dynamic testing process



Testing in the project lifecycle



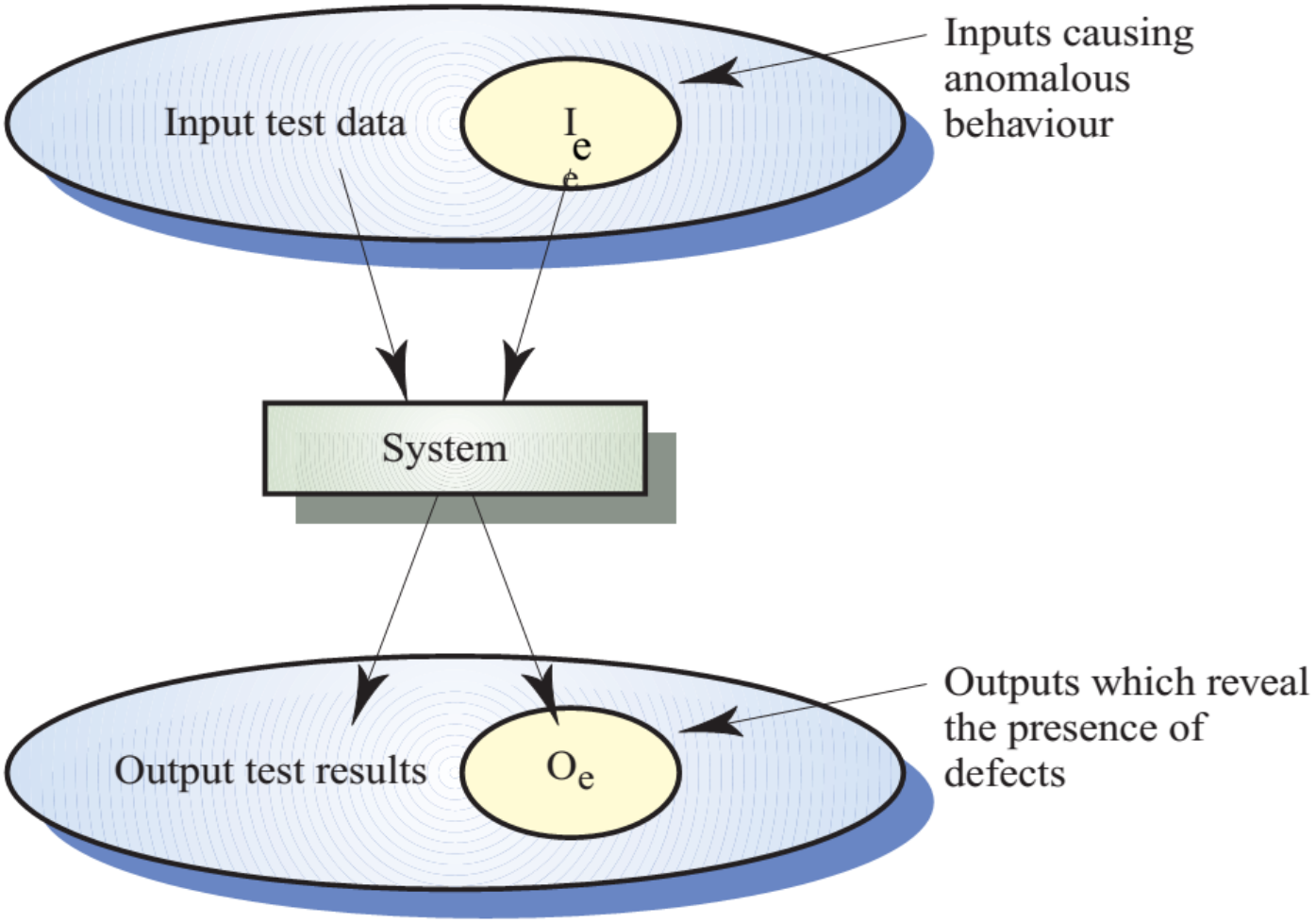
6.2 TECHNIQUES OF TESTING



Black-box Testing

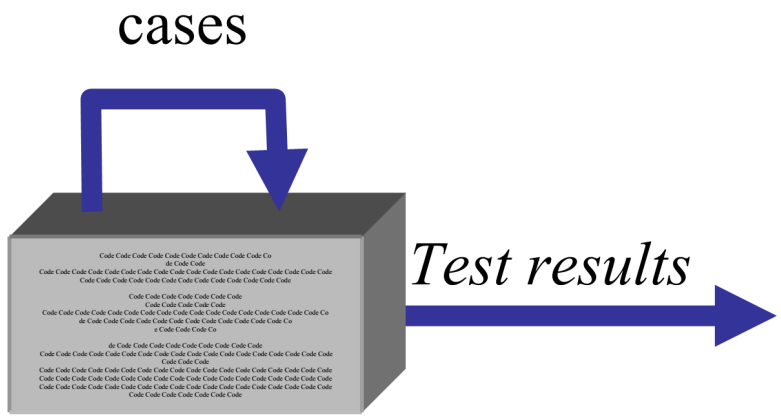
- Approach to testing where the program is considered as a **'black-box'**
- The program test cases are based on the system specification
- Inputs from test data may reveal anomalous outputs, i.e. defects
- Test planning can begin early in the software process
- Main problem - selection of inputs
  - equivalence partitioning

Black-box Testing

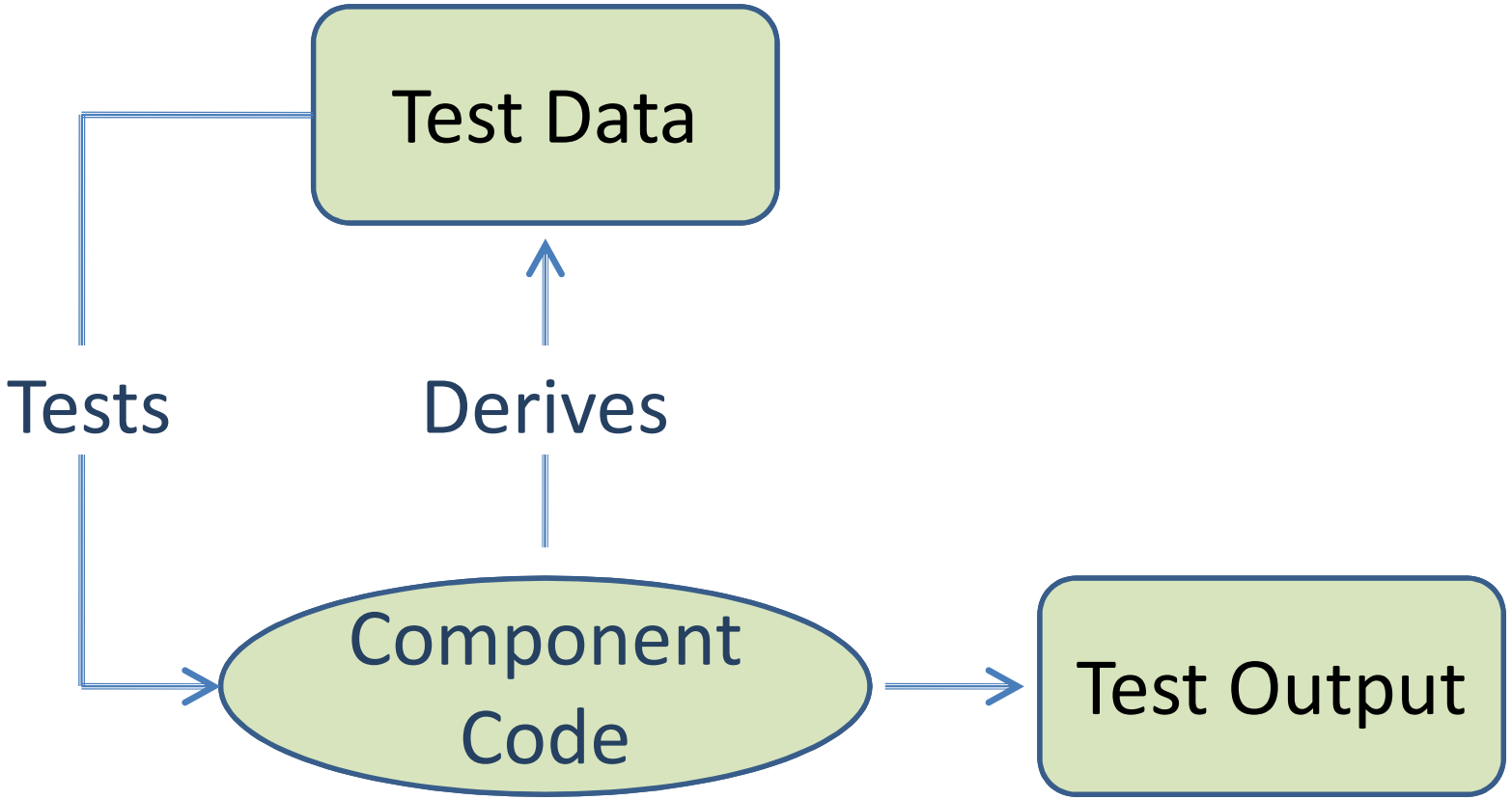


White-box Testing

- Sometimes called Structural testing or glass box testing
- Derivation of test cases according to program structure.
- Knowledge of the program used to identify additional test cases
- Objective is to exercise all program statements (not all path combinations)



Structural testing



Software Inspection

- Inspection techniques include,
  - Program inspections
  - Automated source code analysis
  - Formal verification
- Can only check the correspondence between a program and its specification (verification)
- Unable to demonstrate that the software is operationally useful.
- Can not check non-functional requirements (performance, reliability)

Software Inspection

- Inspections not require execution of a system so may be used before implementation.
- They have been shown to be an effective technique for discovering program errors.
  - Many different defects may be discovered in a single inspection. In testing, one defect may mask another, so several executions are required.
  - The reuse domain and programming knowledge so reviewers are likely to have seen the types of error that commonly arise.

Program Inspections

- Formalized approach to document reviews
- Intended explicitly for defect detection (not correction).
- Defects may be logical errors, anomalies in the code that might indicate an erroneous condition (e.g. an uninitialized variable) or non-compliance with standards.

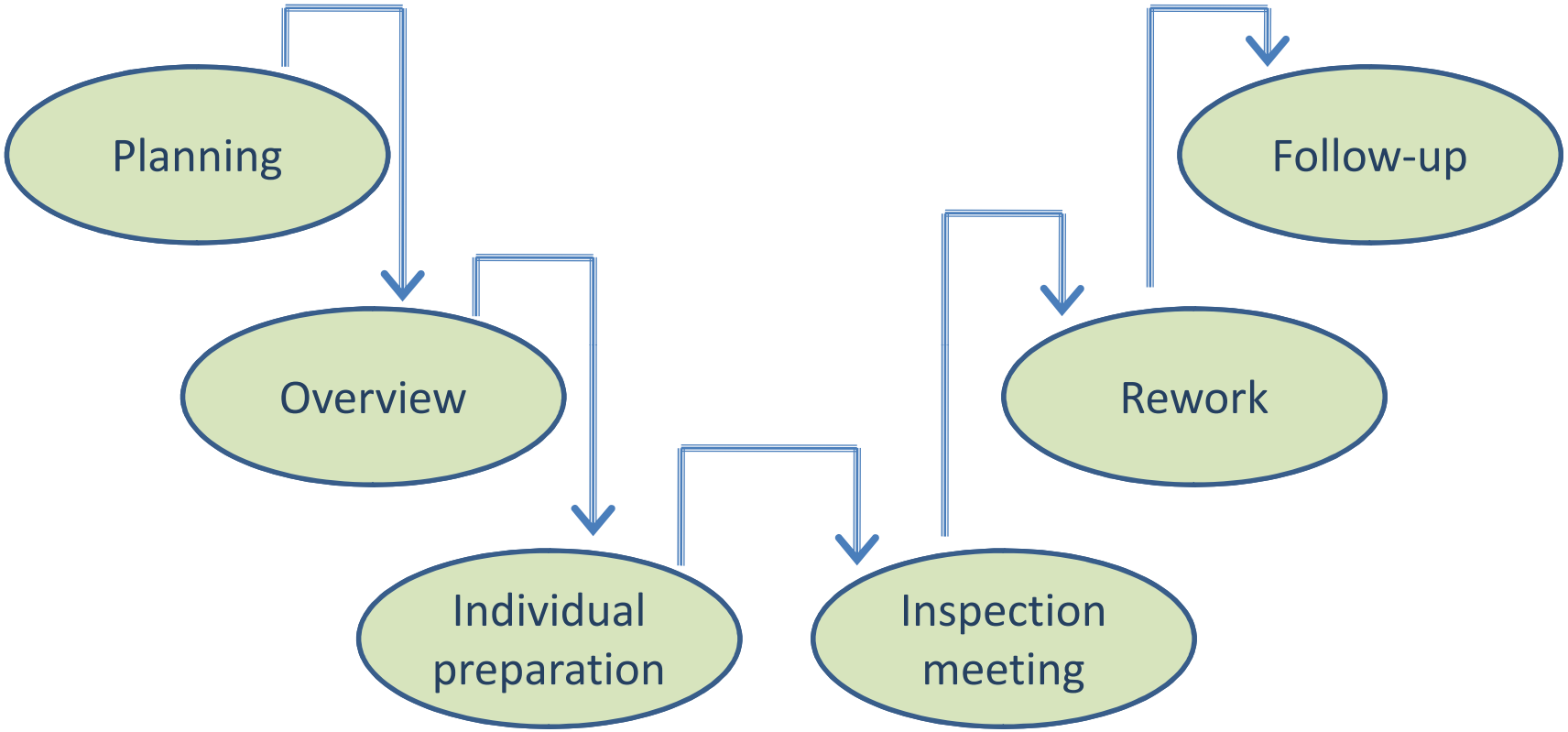
Inspection Roles

Inspector	Role
Programmer or Designer	The programmer or designer responsible for producing the program or document. Responsible for fixing defects discovered during the inspection process.
Inspector	Finds errors, omissions and inconsistencies in programs and documents. May also identify broader issues that are outside the scope of the inspection team.
Reader	Presents the code or document at an inspection meeting.
Scribe	Records the results of the inspection meeting.
Chairman or Moderator	Manages the process and facilitates the inspection. Reports process results to the Chief moderator.
Chief moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

Inspection Pre-conditions

- A precise specification must be available.
- Team members must be familiar with the organization standards.
- Syntactically correct code or other system representations must be available.
- An error checklist should be prepared.
- Management must accept that inspection will increase costs early in the software process.
- Management should not use inspections for staff appraisal i.e. finding out who makes mistakes.

The Inspection Process



The Inspection Process

- System overview presented to inspection team.
- Code and associated documents are distributed to inspection team in advance.
- Inspection takes place and discovered errors are noted.
- Modifications are made to repair discovered errors.
- Re-inspection may or may not be required.

Inspection Checks

Data faults	Are all program variables initialized before their values are used? Have all constants been named? Should the upper bound of arrays be equal to the size of the array or Size -1? If character strings are used, is a delimiter explicitly assigned? Is there any possibility of buffer overflow?
Control faults	For each conditional statement, is the condition correct? Is each loop certain to terminate? Are compound statements correctly bracketed? In case statements, are all possible cases accounted for? If a break is required after each case in case statements, has it been included?
Input/output faults	Are all input variables used? Are all output variables assigned a value before they are output? Can unexpected inputs cause corruption?

Inspection Checks

Interface faults	Do all function and method calls have the correct number of parameters? Do formal and actual parameter types match? Are the parameters in the right order? If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	If a linked structure is modified, have all links been correctly reassigned? If dynamic storage is used, has space been allocated correctly? Is space explicitly de-allocated after it is no longer required?
Exception management faults	Have all possible error conditions been taken into account?

6.3 LEVELS OF TESTING

Test Phases

Test Phase	Test Plan	Author	Technique	Run by
Unit Test	Code design	Designer	White box, Black box, Static	Programmer
Integration Test	Functional specification	Author of specification	Black box, white box, Top-down, bottom-up	Programming team
Interface Test	System Specification	Authors of the system specification	White box	Programmers
System Test	Requirements	Analyst	Black box, stress testing, performance testing	System test team

Test Phases

Test Phase	Test Plan	Author	Technique	Run by
Acceptance Test	Requirements	Analyst / Customer	Black box	Analyst / Customer
Alpha Test	No test plan		Black box	Selected set of users
Beta Test	No test plan		Black box	Any user
Regression Test	Functional specification / Requirements	Analyst	Black box	Development team, System test team

Unit Testing

- Unit Testing is carried out as a part of the coding task. This phase is based on the design of the software for a piece of code. Unit testing should prove the following about the code
  - **Robust** –the code should not fail under any circumstances.
  - **Functionally correct** –the code should carry out the task defined by the code design.
  - **Correct interface** –the inputs and outputs from the code should be as defined in the design.

Unit Test Plan

- The unit test plan must be based on the design of the code and not the code itself. Therefore, the test plan will be written after the completion of design but before the start of the coding phase.
- The test plan is the responsibility of the designer of the code. The testing is usually carried out by the author of the code.

Integration/Sub-systems Testing

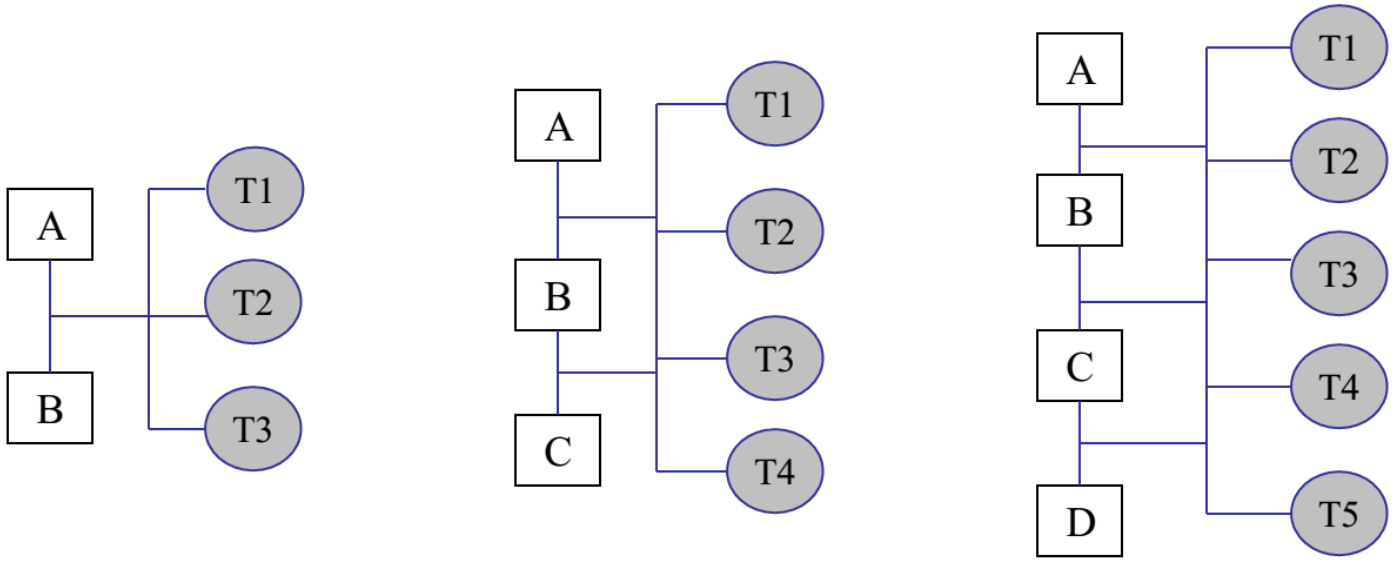
- Integration Testing is carried out after the separate software modules have been unit testing. Integration testing is based on the functional specification of the software. Integration testing should prove the following about the software:
  - **Integration** -the modules of the system should interact as designed.
  - **Functionally correct** -the system should behave as defined in the functional specification.

Integration test plan

- This is based on the specification of the system. The test plan is usually written by the authors of the system specification to avoid any assumptions made during design from being incorporated into the test plan.

Incremental Integration Testing

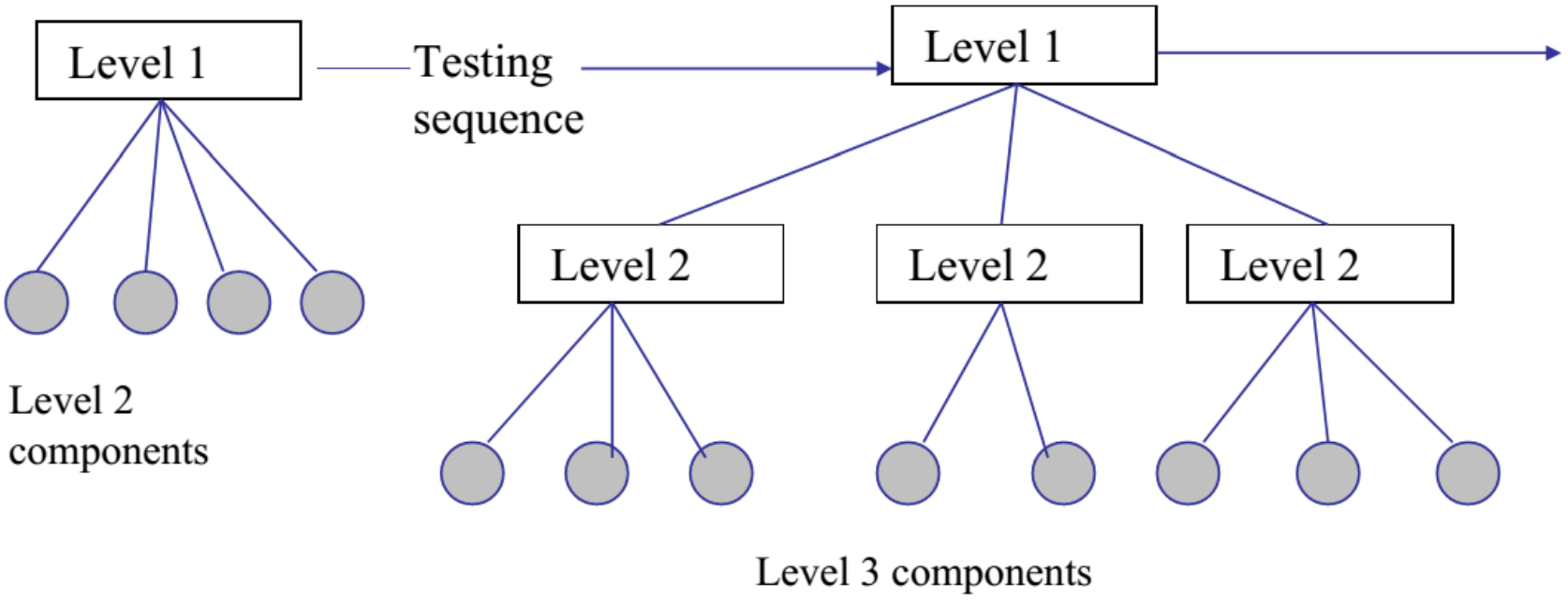
Initially, integrate a minimal system configuration and test the system. Then add components to this minimal configuration and test after each added increment.



T1,T2 and T3 tests are carried out after integration of components A and B. If the tests are successful, add C and carry out tests T1,T2,T3 and T4. If new errors introduced, that is due to the integration of C.

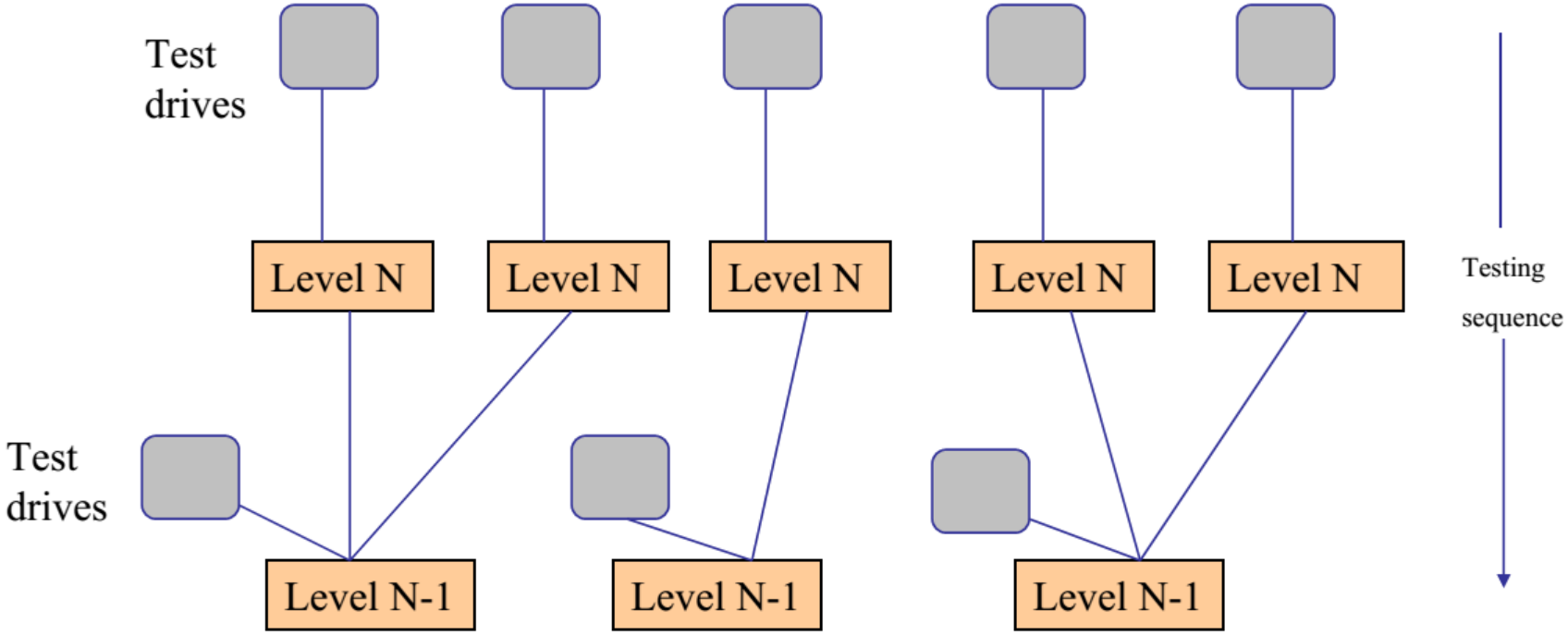
Top-Down Integration Testing

- In top-down integration, the high-level components of a system are integrated and tested before design and implementation of some of the high-level components has been completed.



Bottom-up Integration Testing

- Low-level components are integrated and tested before the higher-level components have been developed.



Interface Testing

- Interface testing takes place when modules or subsystems are integrated to create large systems.
- Each module or sub-system has a defined interface which is called by other program components.
- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
- Particularly important for object-oriented development as objects are defined by their interfaces.
- Cannot be detected by testing the individual objects.
  - The errors will occur due to the interaction between objects.

Types of Interfaces

- **Parameter Interfaces**
  - These are interfaces where the data or sometimes function references are passed from one component to another.
- **Shared memory Interfaces**
  - These are interfaces where block of memory is shared between sub-systems. Data is placed in memory by one sub-system and retrieved from there by other sub-system.
- **Message passing Interfaces**
  - These are interfaces where one subsystem request a service from another sub-system by passing a message to it.
- **Procedural Interfaces**
  - These are interfaces where one sub-system encapsulates a set of procedures which can be called by other subsystem. Objects and abstract data types have this form of interface.

System Testing

- System testing is carried out at the completion of the integration testing. The purpose of system testing is to prove that the software meets the agreed user requirements and works in the target environment. System testing covers both functional and non-functional requirements.

System Test Plan

- The system test plan is based around the agreed requirements. Test plan covers functional requirements and non-functional requirements such as performance. The system test plan is written by the author of the requirements document to avoid assumption introduced during specification.
- The system test plan will also include tests to cover
  - **Recovery –** Force the system to crash and then try to recover to a sensible state.
  - **Security –** Attempt to access the system without the correct authority, or attempt to carry out restricted functions.
  - **Stress –** Attempt to break the system by overloading it.
  - **Performance –** Ensure the system meets the performance requirements.

Acceptance Testing

- Acceptance testing is carried out at the customers site with the customer in attendance. The purpose of the acceptance test is to show to the customer that the software does indeed work. These tests are usually a sub set of the system test.

Acceptance test plan

- This should be agreed with the customer after the requirements for the software have been agreed. Sometimes the customer will write the test plan in which case it must be agreed with the software developers.

Alpha Testing

- Alpha testing is the first real use of the software product. Having completed system testing the product will be given to a small number of selected customers or possible just distributed within the company itself. The software will be used ‘in anger’ and errors reported to the development and maintenance team.

Beta Testing

- After alpha testing has been completed and any changes made a new version of the product will be released to much wider audience. The objective of Beta testing is to iron out the remaining problems with the product before it is put on the general release.

Regression Testing

- Regression testing carried out after any changes are made to a software application. The purpose of regression test is to prove that the change has been made correctly and that the change has not introduced any new errors.
- Regression test plans are usually a sub-set of the integration or systems test plans. It is designed to cover enough functionality to give confidence that the change has not effected any other part of the system.

Back-to-back testing

- Back-to-back testing means that multiple versions of the software are executed on the same test data. If they all agree on the answer, it is assumed that they are all correct.

Thread testing

- Based on testing an operation which involves a sequence of processing steps which thread their way through the system
- Start with single event threads then go on to multiple event threads
  - **Advantages**
    - Suitable for real-time and object-oriented systems
  - **Disadvantage**
    - Difficult to assess test adequacy, because of large number of event combinations

Statistical Software Testing

- Statistical testing involves exercising a program with generated random inputs, the test profile and the number of generated inputs being determined according to criteria based on program structure or software functionality.
- In case of complex programs, the probabilistic generation must be based on a black box analysis, the adopted criteria being defined from behavior models deduced from the specification.

Object Oriented Testing

In an object oriented system, four levels of testing can be identified:

1. **Testing the individual operations associated with objects** – These are functions or procedures and the black-box and white-box approaches may be used.
2. **Testing individual object classes** – The principle of black-box testing is unchanged but the notion of an equivalence class must be extended to cover related operation sequences.
3. **Testing clusters of objects** – Strict top-down or bottom-up integration may be inappropriate to create groups of related objects. Other approaches such as scenario based testing should be used.
4. **Testing the object oriented system** – V & V against the systems requirements specification is carried out in exactly the same way as for any other type of system.

Object Class Testing

- When testing objects, complete test coverage should include:
  1. the testing in isolation of all operations associated with the object;
  2. the setting and interrogation of all attributes associated with the object;
  3. the exercise of the object in all possible states. This means that all events that cause a state change in the object should be simulated.

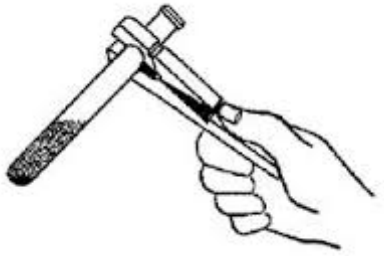
Object Integration

- In object oriented systems, there is no obvious ‘top’ that provides for the integration nor is there a clear hierarchy of objects that can be created.
- Clusters therefore have to be created using knowledge of there operation and the features of the system that are implemented by these clusters.

Object Integration

- There are three possible approaches to integration testing that may be used:
  1. **Use-case or scenario-based testing** – Testing can be based on the scenario descriptions and object clusters created to support the use-cases that relate to that mode of use.
  2. **Thread testing** – Thread testing is based on testing the system's response to a particular input or set of input events.
  3. **Object interaction testing** – An intermediate level of integration testing can be based on identifying 'method – message ' paths. These are traces through a sequence of object interactions which stop when an object operation does not call on the services of any other object.

6.4 DESIGN OF TEST CASES



Design of Test Cases

- Test case design is a part of system and component testing.
- The objective of the test case designing process is to develop test cases.
- These test cases are used to detect program defects and also it detects whether the system meets its requirements.

Design of Test Cases

- When designing a test case,
  - select a particular feature of the system.
  - Select set of inputs that execute that feature.
  - Decide possible outputs of the test case.
  - Check the difference between actual and the expected out comes after the testing procedure.

Test case designing approaches

- **Requirements-based testing -**Test cases are designed to test the system requirements. Mostly used at the system testing stage. Test cases should be created for each requirement to verify that the system meets requirements.
- **Partition testing -** Identify input and output partitions. The system execute inputs from all partitions.
- **Structural testing -** Use the knowledge of program structure to design tests that exercise all parts of the program. When testing a program, each statement should be execute at least once.

6.5 QUALITY MANAGEMENT ACTIVITIES

Quality Management

- Software quality management can be structured into three principal activities:

1. **Quality assurance**
   - The establishment of a framework of organizational procedures and standards which lead to high quality software.
2. **Quality planning**
   - The selection of appropriate procedures and standards from this framework and adaptation of these for a specific software project.
3. **Quality control**
   - The definition and enactment of processes which ensure that the project quality procedures and standards are followed by the software development team.

Software Quality Management

- Quality management provides an independent check on the software development process. The deliverables from the software process are input to the quality management process and are checked to ensure that they are consistent with organizational standards and goals.
- Quality management should be separated from project management so that quality is not compromised by management responsibilities for project budget and schedule.
- An independent team should responsible for quality management and should report to the management above the project management level.

Software Quality Management

- There must be a management commitment to quality.
- Concerned with ensuring that the required level of quality is achieved in a software product.
- Involves defining appropriate quality standards and procedures and ensuring that these are followed.
- Should aim to develop a ‘quality culture’ where quality is seen as everyone’s responsibility.

What is Quality?

- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
  - Some quality requirements are difficult to specify in an unambiguous way;
  - Software specifications are usually incomplete and often inconsistent.

What is Quality?

- **Traditional view**
  - quality is about perfection/bug free code. Generally associated with testing at the end of development. Testing cannot introduce quality to a product, it can only reduce the number of defects in the product.
- **Modern view (ISO 9000)**
  - Good quality is not perfection but fit for the purpose. Build the right product in the right way. Do not over engineer (too expensive). Do not under engineer (not fit for the purpose).

Scope of Quality Management

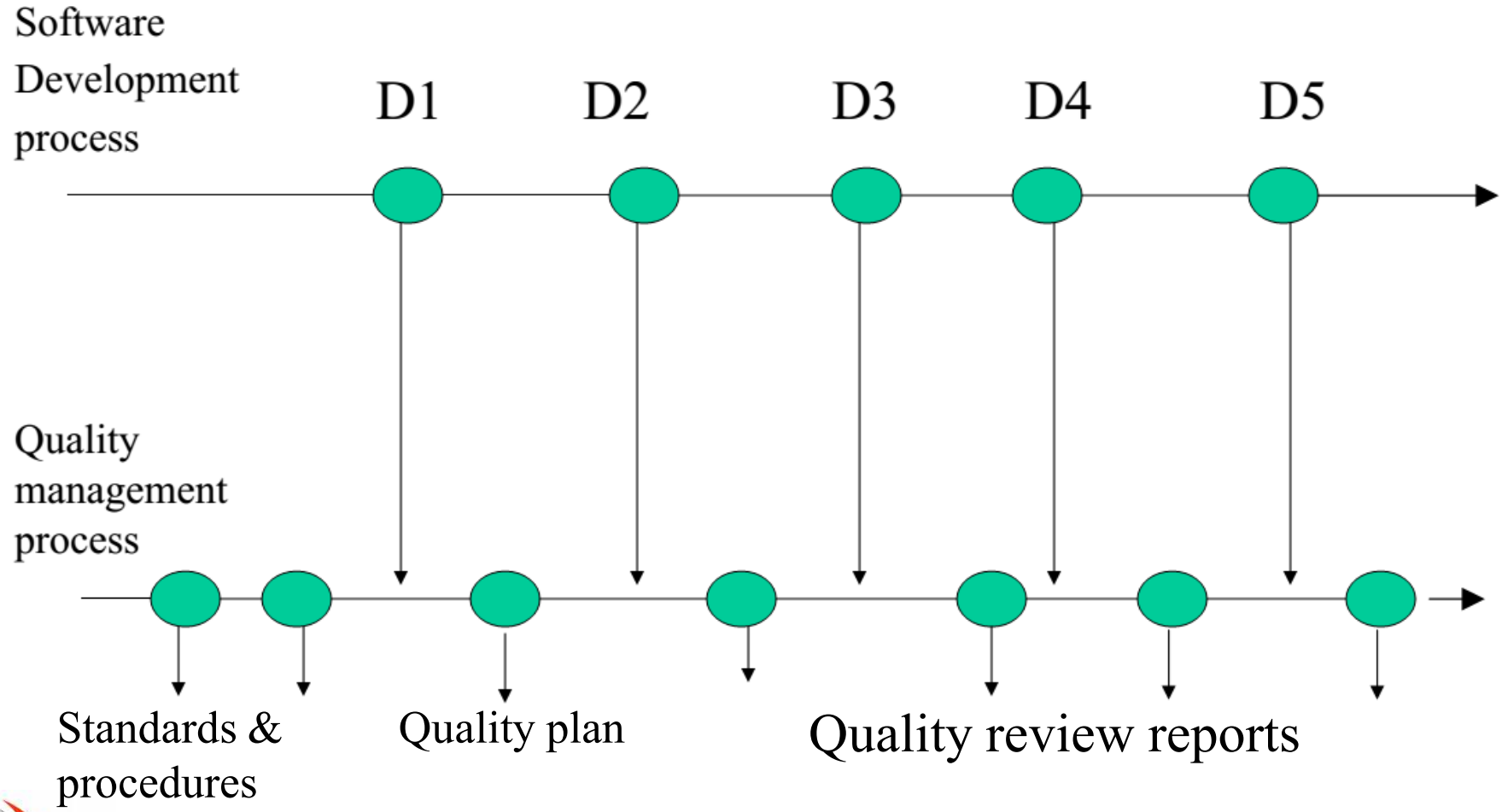
- Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.
- For smaller systems, quality management needs less documentation and should focus on establishing a quality culture.

Quality Management Activities

- **Quality assurance**
  - Establish organizational procedures and standards for quality.
- **Quality planning**
  - Select applicable procedures and standards for a particular project and modify these as required.
- **Quality control**
  - Ensure that procedures and standards are followed by the software development team.

Quality management should be separate from project management to ensure independence.

Quality Management and Software Development



6.6 PRODUCT AND PROCESS QUALITY

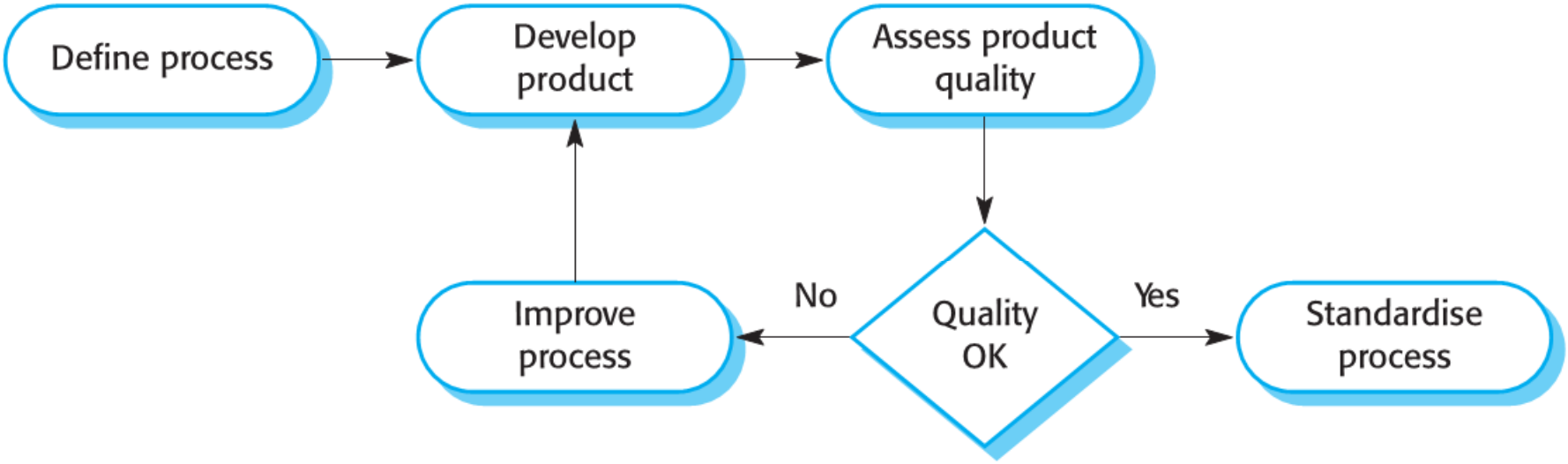
Process and Product Quality

- The quality of a developed product is influenced by the quality of the production process.
- This is important in software development as some product quality attributes are hard to assess.
- However, there is a very complex and poorly understood relationship between software processes and product quality.

Process-based Quality

- There is a straightforward link between process and product in manufactured goods.
- More complex for software because:
  - The application of individual skills and experience is particularly important in software development;
  - External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality.
- Care must be taken not to impose inappropriate process standards - these could reduce rather than improve the product quality.

Process-based Quality



Practical Process Quality

- Define process standards such as how reviews should be conducted, configuration management, etc.
- Monitor the development process to ensure that standards are being followed.
- Report on the process to project management and software procurer.
- Don’t use inappropriate practices simply because standards have been established.

6.7 STANDARDS

Quality Assurance and Standards

- Quality assurance(QA) activities define a framework for achieving software quality.
- There are two types of standards that may be established as a part of the quality assurance process:
- ***Product standards***
  - These are standards that apply to the software product being developed. They include standards such as document standards (eg. Specification of a class), coding standards and user interface standards. Product quality includes reusability, usability, portability, maintainability etc.
- ***Process standards***
  - These are standards that define the processes which should be followed during software development. They may include definitions of specification, design and validation processes and a description of the documents which must be generated in the course of these processes.

Importance of Standards

- Encapsulation of best practice
  - avoids repetition of past mistakes.
- They are a framework for quality assurance processes
  - they involve checking compliance to standards.
- They provide continuity
  - new staff can understand the organization by understanding the standards that are used.

Product and Process Standards

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of documents to CM
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

Problems with Standards

- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious manual work is often involved to maintain the documentation associated with the standards.

ISO9000

- ISO9000 is a set of international standards that can be applied to a range of organizations from manufacturing through to service industries.
- ISO9001 is the most general of these standards and applies to organizations concerned with quality processes regarding design, development and maintain products. This is a generic model of a quality process which describes various aspects of that process and defines which standards and procedures could exist within an organization.
- ISO 9000-3 interprets ISO 9000 for software development.

ISO9000

- ISO900-1 is a general guideline which gives background information about the family of standards.
- ISO9002 and ISO9003 are subsets of ISO9001.
- ISO9002 is used for situations in which there is no design.
- ISO9003 is used for situations in which there is neither design nor production (eg. Retail).
- ISO9000-3 is a guideline on how to use ISO9001 for software development.

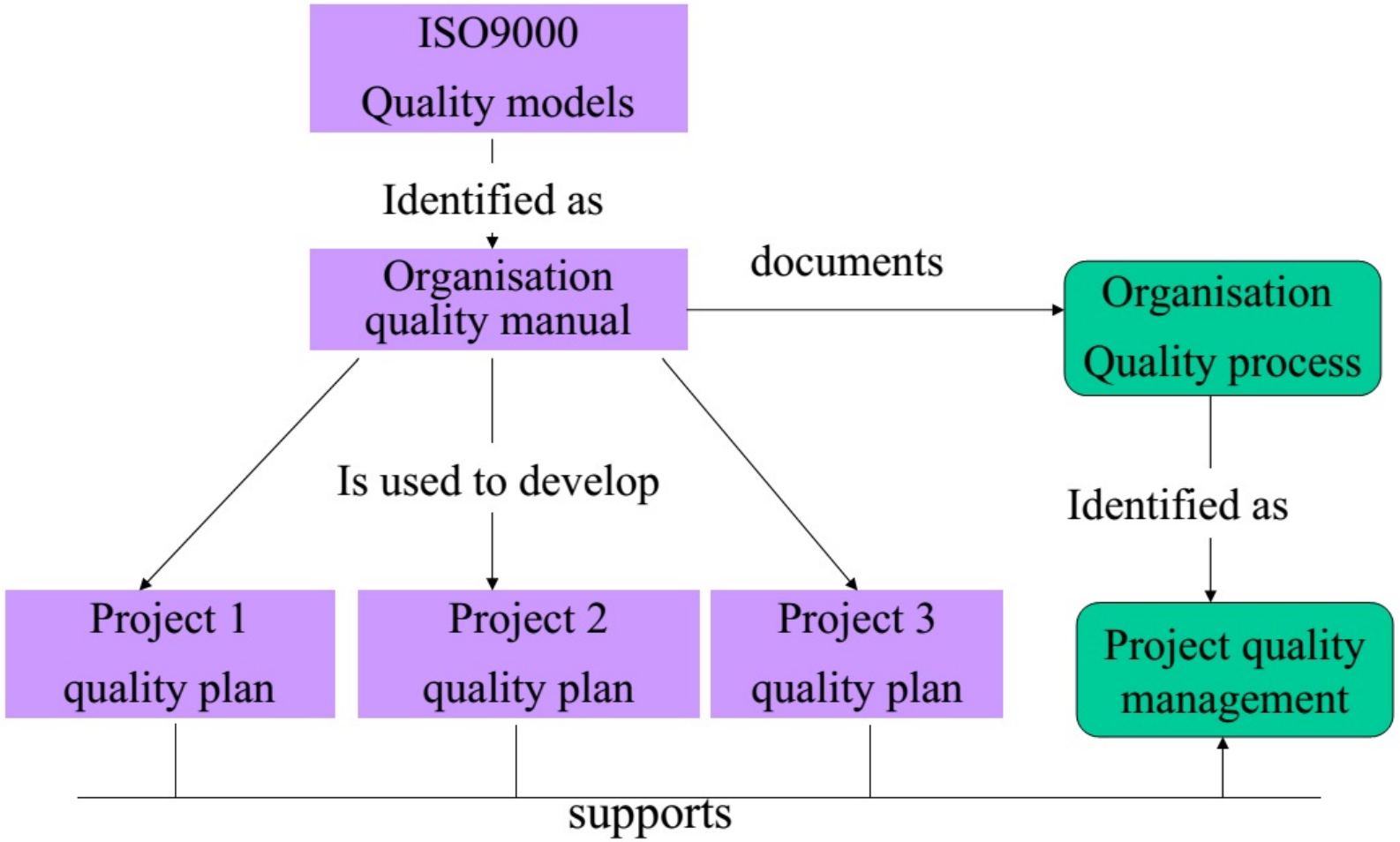
ISO9000

- When asked whether cost, timescale or quality was most important when completing against another company for a software project *Tomoo Mastubara of Hitachi Software Engineering* replied

  "Quality is first! Always first. If we deliver bad quality to the customer, the customer will complain many times, over and over. But if we are late, he will complain only once and then may be he will forget. And if we have underestimated the cost, the customer will not complain at all. – for he will know we have made the mistake. WE will bear the burden of the cost mistake"

- Any serious quality initiative will repay any set-up costs within the first years through improved productivity and customer satisfaction.

ISO9000 and Quality Management

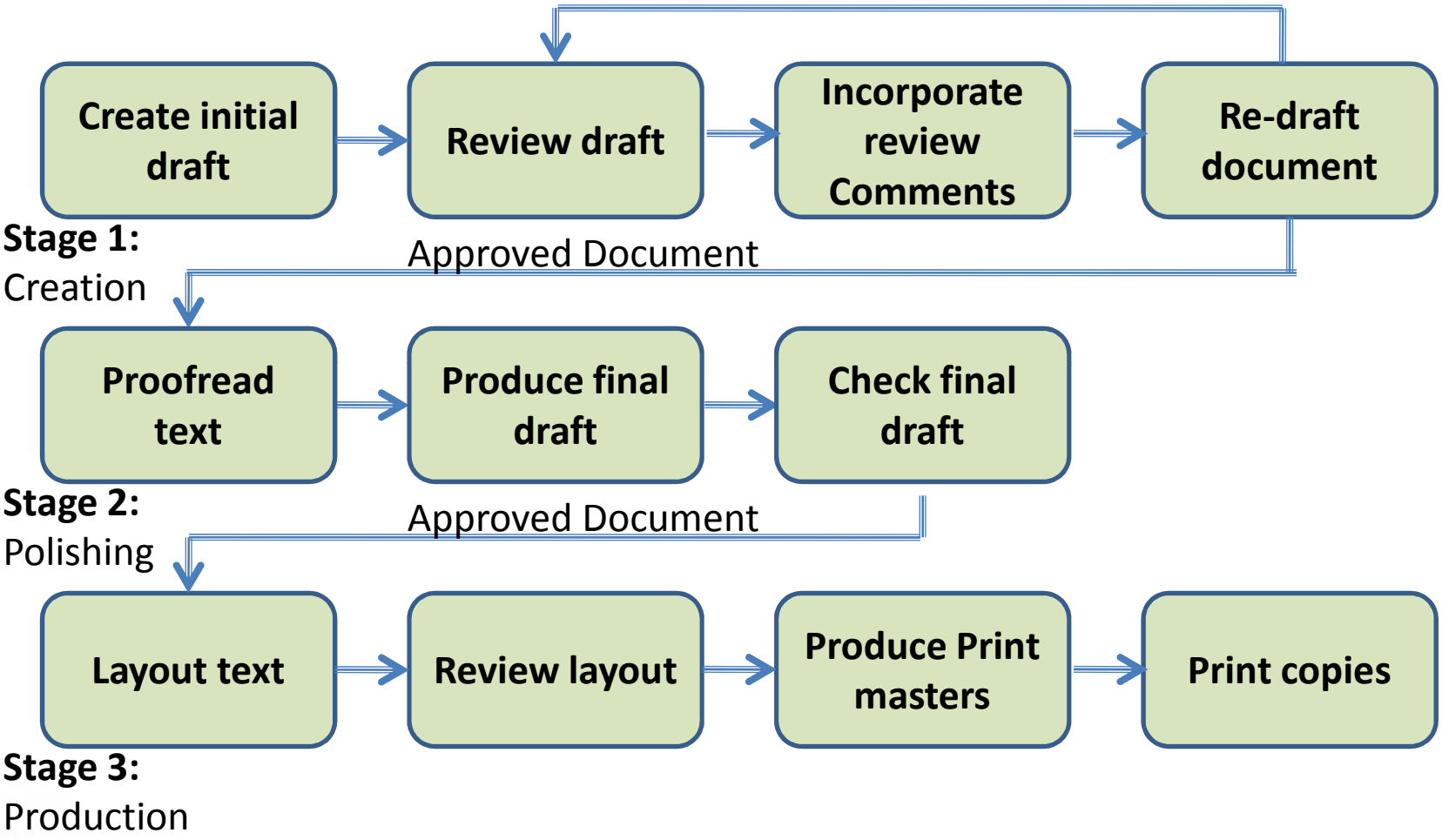


Documentation Standards

- Documentation standards in a software project are particularly important as documents are the only tangible way of representing the software and the software process.

- There are three types of documentation standards
  1. **Documentation Process Standards**
     - Define the process which should be followed for document production.
  2. **Document Standards**
     - Govern the structure and presentation of documents.
  3. **Document Interchange Standards**
     - Ensure that all electronic copies of documents are compatible.

A document production process including quality checks



Capability Maturity Model (CMM)

- CMM was developed at the Software Engineering Institute (SEI) in Pittsburgh, and it is very much a rival to ISO9001 for software.
- CMM Is a scheme to classify a software development organization according to its capability. CMM identifies five different maturity levels for software developing organizations.

Capability Maturity Model (CMM)

1. **Initial** - The software development is run informally, and depends on the competence of some persons.
2. **Repeatable** - There is a common system for project management and control.
3. **Defined** - There is a common system for the software engineering activities.
4. **Managed** - The software development process is stable and gives a consistent product quality. Measurements are used to keep the process and product under control.
5. **Optimizing** - The software development process contain its own improvement process. Process improvement is budgeted and planned and is an integral part of the organization's process.

Key Points

- Software quality management is concerned with ensuring that software has a low number of defects and that it reaches the required standards of maintainability, reliability, portability and so on
- SQM includes defining standards for processes and products and establishing processes to check that these standards have been followed

Key Points

- Software standards are important for quality assurance as they represent an identification of ‘best practice’
- Quality management procedures may be documented in an organizational quality manual, based on the generic model for a quality manual suggested in the ISO 9001 standard
- Product quality metrics are particularly useful for highlighting anomalous components that may have quality problems