Software Verification

Background

- Main objectives of a project: High Quality & High Productivity (Q&P)
- Quality has many dimensions
  - reliability, maintainability, interoperability etc.
- Reliability is perhaps the most important
- Reliability: The chances of software failing
- More defects => more chances of failure => lesser reliability
- Hence Quality goal: Have as few defects as possible in the delivered software

Faults & Failure

- Failure: A software failure occurs if the behavior of the s/w is different from expected/specified.
- Fault: cause of software failure
- Fault = bug = defect = error
- Failure implies presence of defects
- A defect has the potential to cause failure.
- Definition of a defect is environment, project specific

Verification vs. validation

- Verification: "Are we building the product right"
  - The software should conform to its specification
- Validation: "Are we building the right product"
  - The software should do what the user really requires
- V & V must be applied at each stage in the software process
- Two principal objectives
  - Discovery of defects in a system
  - Assessment of whether the system is usable in an operational situation

Static and dynamic verification

- *STATIC – Software inspections*
  - Concerned with analysis of the static system representation to discover problems
  - May be supplement by tool-based document and code analysis
- *DYNAMIC – Software testing*
  - Concerned with exercising and observing product behaviour
  - The system is executed with test data and its operational behaviour is observed

Program testing

- Can reveal the presence of errors, not their absence
- A successful test is a test which discovers one or more errors
- The only validation technique for non-functional requirements
- Should be used in conjunction with static verification to provide full V&V coverage

Types of testing

- Defect testing
  - Tests designed to discover system defects.
  - A successful defect test is one which reveals the presence of defects in a system.
- Statistical testing
  - Tests designed to reflect the frequency of user inputs
  - Used for reliability estimation

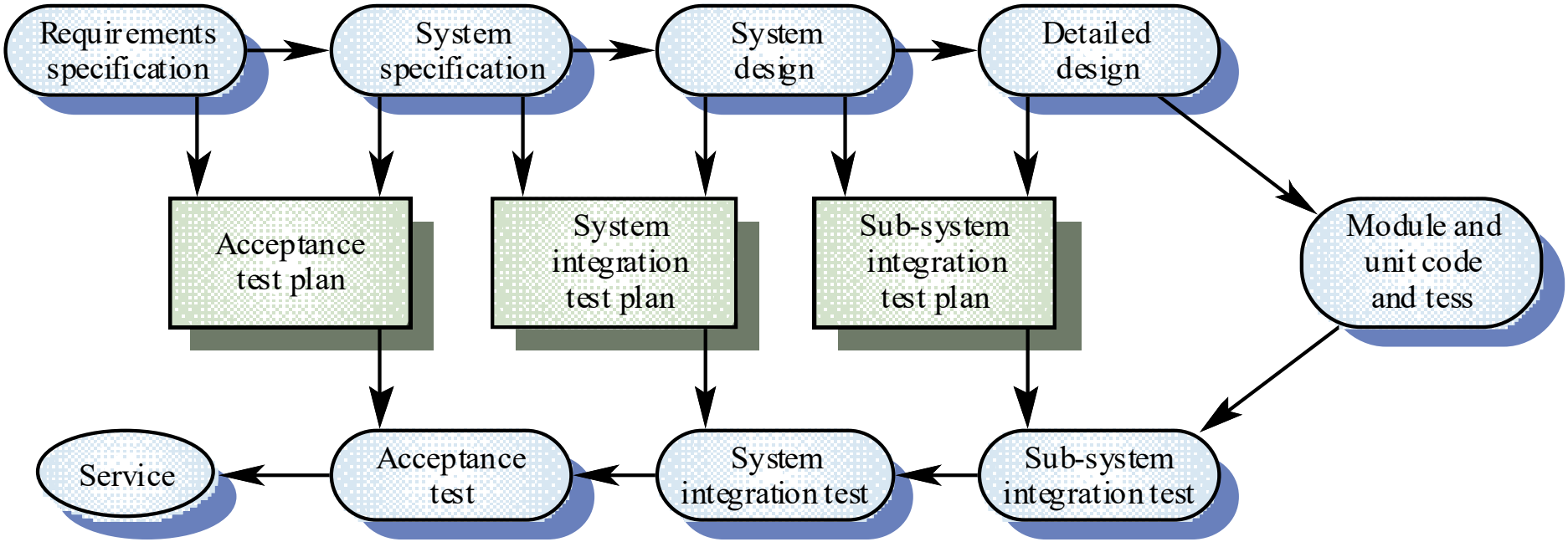
Testing and debugging

- Defect testing and debugging are distinct processes
- Verification and validation is concerned with *establishing the existence* of defects in a program
- Debugging is concerned with *locating and repairing* these errors
  - Debugging involves formulating hypotheses about program behaviour then testing these hypotheses to find the system error

V & V planning

- Careful planning is required to get the most out of testing and inspection processes
- Planning should start early in the development process
- The plan should identify the balance between static verification and testing
- Test planning is about defining standards for the testing process rather than describing product tests

The V-model of development



Software inspections

- Involve people examining the source representation with the aim of discovering anomalies and defects
- Do not require execution of a system
  - May be used before implementation
- May be applied to any representation of the system
  - Requirements, design, test data, etc.
- Very effective technique for discovering errors
- Many different defects may be discovered in a single inspection
  - In testing, one defect may mask another so several executions are required
- Reuse of domain and programming knowledge
  - Reviewers are likely to have seen the types of error that commonly arise

Inspections and testing

- Inspections and testing are complementary and not opposing verification techniques
- Both should be used during the V & V process
- Inspections can check conformance with a specification but not conformance with the customer's real requirements
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

Program inspections

- Formalised 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)
  - non-compliance with standards

Inspection procedure

- 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 teams

- Made up of at least 4 members
- Author of the code being inspected
- Inspector who finds errors, omissions and inconsistencies
- Reader who reads the code to the team
- Moderator who chairs the meeting and notes discovered errors
- Other roles are Scribe and Chief moderator

Inspection checklists

- Checklist of common errors should be used to drive the inspection
- Error checklist is programming language dependent
- The 'weaker' the type checking, the larger the checklist
- Examples
  - Initialisation
  - Constant naming
  - Loop termination
  - Array bounds

Inspection rate

- 500 statements/hour during overview
- 125 source statement/hour during individual preparation
- 90-125 statements/hour can be inspected
- Inspection is therefore an expensive process
- Inspecting 500 lines costs about 40 person/hours

Automated static analysis

- Static analysers are software tools for source text processing
- They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team
- Very effective as an aid to inspections. A supplement to but not a replacement for inspections

Static analysis checks

Fault class	Static analysis check
Data faults	Variables used before initialisation Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic

Stages of static analysis (1)

- *Control flow analysis*
  - Checks for loops with multiple exit or entry points, finds unreachable code, etc.
- *Data use analysis*
  - Detects uninitialized variables, variables written twice without an intervening assignment, variables which are declared but never used, etc.
- *Interface analysis*
  - Checks the consistency of routine and procedure declarations and their use

Stages of static analysis (2)

- *Information flow analysis*
  - Identifies the dependencies of output variables
  - Does not detect anomalies, but highlights information for code inspection or review
- *Path analysis*
  - Identifies paths through the program and sets out the statements executed in that path
  - Also potentially useful in the review process
- Both these stages generate vast amounts of information
  - Handle with caution!

Experience with inspection

- Raytheon
  - Reduced "rework" from 41% of cost to 20% of cost
  - Reduced effort to fix integration problems by 80%
- Paulk et al.: cost to fix a defect in space shuttle software
  - $1 if found in inspection
  - $13 during system test
  - $92 after delivery
- IBM
  - 1 hour of inspection saved 20 hours of testing
  - Saved 82 hours of rework if defects in released product
- IBM Santa Teresa Lab
  - 3.5 hours to find bug with inspection, 15-25 through testing
- C. Jones
  - Design/code inspections remove 50-70% of defects
  - Testing removes 35%
- R. Grady, efficiency data from HP
  - System use 0.21 defects/hour
  - Black box 0.28 defects/hour
  - White box 0.32 defects/hour
  - Reading/inspect 1.06 defects/hour

- Your mileage may vary
  - Studies give different answers
  - These results show what is possible

Kinds of Inspections

Inspections / Formal Technical Reviews

- Participation defined by policy
  - Developers
  - Designated key individuals – peers, QA team, Review Board, etc.
- Advance preparation by participants
  - Typically based on checklists
- Formal meeting to discuss artifact
  - Led by moderator, not author
  - Documented process followed
  - May be virtual or conferenced
- Formal follow-up process
  - Written deliverable from review
  - Appraise product

Walkthroughs

- No advance preparation
- Author leads discussion in meeting
- No formal follow-up
- Low cost, valuable for education

Other review approaches

- Pass-around – preparation part of an inspection
- Peer desk check – examination by a single reviewer (like pair programming)
- Ad-hoc – informal feedback from a team member

There are tradeoffs among the techniques

- Formal technical reviews will find more bugs
  - Ford Motor: 50% more bugs with formal process
- But they also cost more

Review Roles

Moderator

- Organizes review
  - Keeps discussion on track
  - Ensures follow-up happens
- Key characteristics
  - Good facilitator
  - Knowledgeable
  - Impartial and respected
  - Can hold participants accountable and correct inappropriate behavior
- Separate role from **Recorder**
  - Who captures a log of the inspection process

Review Roles

Reader (different from author)

- Presents material
  - Provides points of comparison for author and other team members
    - Differences in interpretation provoke discussion
    - Reveals ambiguities
      - If author were to present, others might not mention that their interpretation is different
- Alternative
  - Get comments section by section
  - Faster, but does not capture differing perspectives as effectively

Review Roles

Author

- Describes rationale for work
- Not moderator or reader
  - Conflict between objectivity required of moderator/reader and advocacy for the author's own work
  - Others raise issues more comfortably
- Not recorder
  - Temptation to not write down issues the author disagrees with
- Significant benefits to attending
  - Gain insight from others' perspectives
  - Can answer questions
  - Can contribute to discussion based on knowledge of artifact
  - Potential downside: meeting may be confrontational

Case Study: Example

1. Inspection process model: Example
2. Inspection Moderator checklist: Example
3. Java Checklist: Example
4. Inspection Typo Log: Example
5. Inspection Issue Log: Example
6. Inspection Summary Report: Example
7. Inspection Lesson Learned Questions: Example

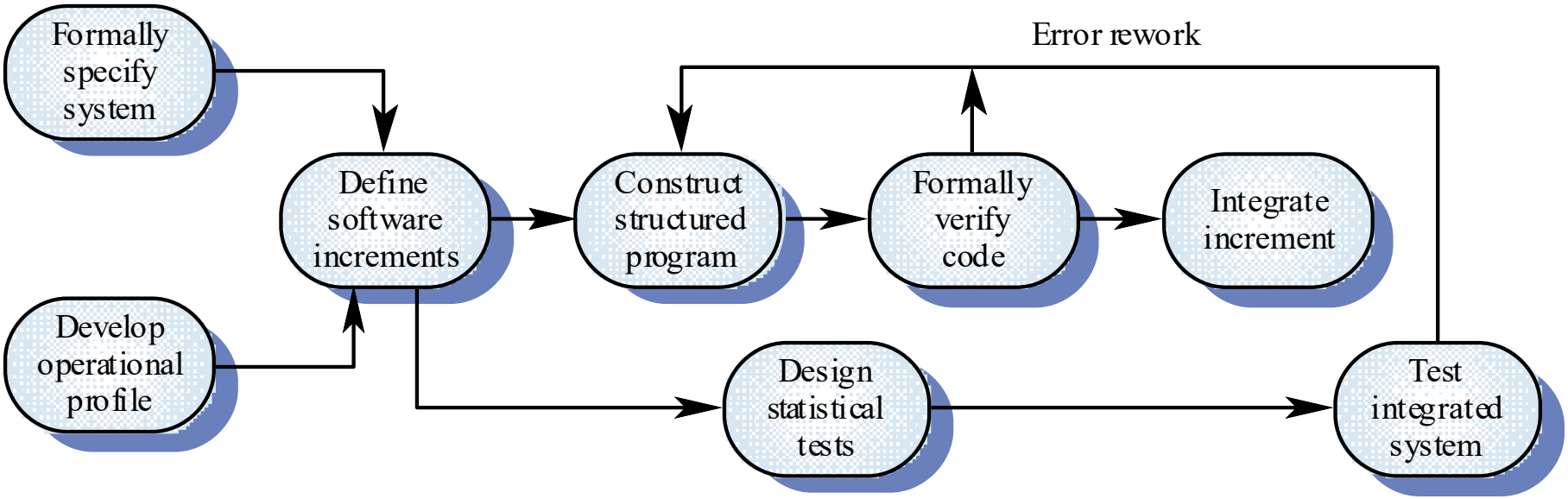
Key points

- Verification and validation are not the same thing.
  - Verification shows conformance with specification
  - Validation shows that the program meets the customer's needs
- Test plans should be drawn up to guide the testing process
- Static verification techniques involve examination and analysis of the program for error detection
- Program inspections are very effective in discovering errors
- Program code in inspections is checked by a small team to locate software faults
- Static analysis tools can discover program anomalies which may be an indication of faults in the code
- The Cleanroom development process depends on incremental development, static verification and statistical testing

Cleanroom software development

- The name is derived from the 'Cleanroom' process in semiconductor fabrication. The philosophy is defect avoidance rather than defect removal
- Software development process based on:
  - Incremental development
  - Formal specification
  - Static verification using correctness arguments
  - Statistical testing to determine program reliability

The Cleanroom process



Cleanroom software development

- Following tasks occur in cleanroom engineering:

1. Incremental planning
   - In this task, the incremental plan is developed.
   - The functionality of each increment, projected size of the increment and the cleanroom development schedule is created.
   - The care is to be taken that each increment is certified and integrated in proper time according to the plan.
2. Requirements gathering
   - Requirement gathering is done using the traditional techniques like analysis, design, code, test and debug.
   - A more detailed description of the customer level requirement is developed.

Cleanroom software development

3. Box structure specification
   - The specification method uses box structure.
   - Box structure is used to describe the functional specification.
   - The box structure separate and isolate the behaviour, data and procedure in each increment.
4. Formal design
   - The cleanroom design is a natural specification by using the black box structure approach.
   - The specification is called as state boxes and the component level diagram called as the clear boxes.

Cleanroom software development

5. Correctness verification
   - The cleanroom conducts the exact correctness verification activities on the design and then the code.
   - Verification starts with the highest level testing box structure and then moves toward the design detail and code.
   - The first level of correctness takes place by applying a set of 'correcting questions'.
   - More mathematical or formal methods are used for verification if correctness does not signify that the specification is correct.
6. Code generation, inspection and verification
   - The box structure specification is represented in a specialized language and these are translated into the appropriate programming language.
   - Use the technical reviews for the syntactic correctness of the code.

Cleanroom software development

7. Statistical test planning
   - Analyzed, planned and designed the projected usages of the software.
   - The cleanroom activity is organized in parallel with specification, verification and code generation.
8. Statistical use testing
   - The exhaustive testing of computer software is impossible. It is compulsory to design limited number of test cases.
   - Statistical use technique execute a set of tests derived from a statistical sample in all possible program executions.
   - These samples are collected from the users from a targeted population.
9. Certification
   - After the verification, inspection and correctness of all errors, the increments are certified and ready for integration.

Cleanroom software development

- Cleanroom process model
  - The modeling approach in cleanroom software engineering uses a method called box structure specification.
  - A 'box' contains the system or the aspect of the system in detail.
  - The information in each box specification is sufficient to define its refinement without depending on the implementation of other boxes.
- The cleanroom process model uses three types of boxes as follows:
- Black box
  - The black box identifies the behavior of a system.
  - The system responds to specific events by applying the set of transition rules.
  - Specifies system function by mapping all possible stimulus histories to all possible responses
  - **S* ->R**

Cleanroom software development

- State box
  - The box consist of state data or operations that are similar to the objects.
  - The state box represents the history of the black box i.e the data contained in the state box must be maintained in all transitions.

S x T -> R x T

stimuli X state data **->** responses X state data

Cleanroom software development

- Clear box
  - The transition function used by the state box is defined in the clear box.
  - It simply states that a clear box includes the procedural design for the state box.
  - Contains the procedural design of the state box, in a manner similar to structured programming
  - Specifies both data flow and control flow

Cleanroom process teams

- Specification team
  - Responsible for developing and maintaining the system specification
- Development team
  - Responsible for developing and verifying the software
  - The software is not executed or even compiled during this process
- Certification team
  - Responsible for developing a set of statistical tests to exercise the software after development
  - Reliability models are used to determine when reliability is acceptable

Cleanroom process evaluation

- Results in IBM have been very impressive with few discovered faults in delivered systems
- Independent assessment shows that the process is no more expensive than other approaches
- Fewer errors than in a 'traditional' development process
- Not clear how this approach can be transferred to an environment with less skilled or less highly motivated engineers

Cleanroom is Shift in Practice

- From
  - Individual craftsmanship
  - Sequential development
  - Individual unit testing
  - Informal coverage testing
  - Unknown reliability
  - Informal design
- To
  - Peer reviewed engineering
  - Incremental development
  - Team correctness verification
  - Statistical usage testing
  - Measured reliability
  - Disciplined engineering specification and design

Benefits

- Zero failures in the field
  - that's the goal any way
  - a realistic expectation is < 5 failures per KLOC on first program execution in the first team project
- Short development cycles
  - results from use incremental strategy and avoidance of rework
  - new teams should experience a two-fold productivity increase on the first project and continue the increase
- Longer product life
  - investments detailed specifications and usage models help keep a product viable longer

Why are Cleanroom Techniques Not Widely Used

- Some people believe cleanroom techniques are too theoretical, too mathematical, and too radical for use in real software development
- Relies on correctness verification and statistical quality control rather than unit testing (a major departure from traditional software development)
- Organizations operating at the ad hoc level of the Capability Maturity Model, do not make rigorous use of the defined processes needed in all phases of the software life cycle