Software Engineering COMP1035

Lecture 15

Software Quality

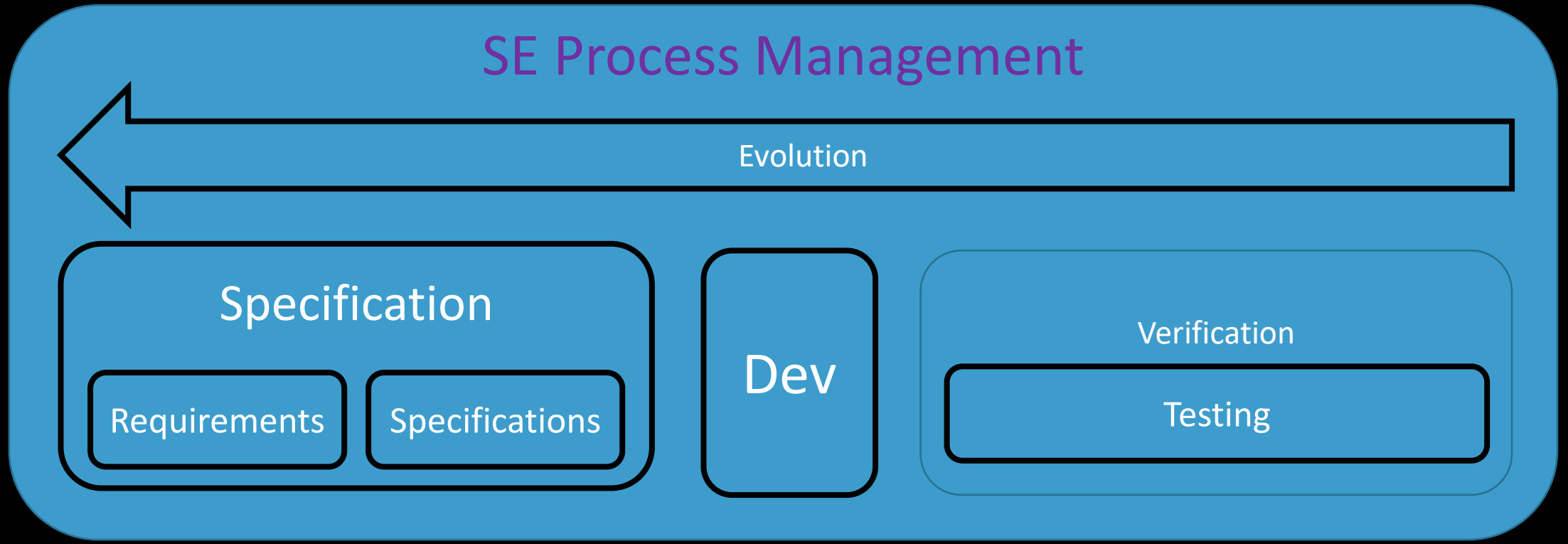
Outline

- Software Quality Management
  1. Planning for quality
  2. Defining standards
  3. Checking for quality

Where We Are In the Process

University of Nottingham
UK | CHINA | MALAYSIA

Keeping Track of SE Module



Process/Project Management

- Deliver good quality product
- Use good reliable methods
- Reducing as much risk as possible
- Plan / budget / manage a project based on these

Process/Project Management

- **Deliver good quality product**
- Use good reliable methods
- Reducing as much risk as possible
- Plan / budget / manage a project based on these

Software Quality

- Most people presume Software Quality is a stage:
  - Release/Acceptance testing stage
- It's not!
- If Quality Management is pervasive - and a "culture"
  - Then the software won't fail Release/Acceptance Testing (as often)
- You can take some measures of quality
  - But they only indicate characteristics

Software Quality Management

- Software Quality means **doing every aspect well**
- To do Release/Acceptance Testing well
  - You need to have done Reqs & Specs well
  - Otherwise the dev team don't know what the goals are
- To do Unit Testing well
  - You need clear specifications and models
- To do Coding well
  - You need agreement on coding standards

Software Quality Management

- Software Engineering Processes were introduced in the 1960s
  - Because software was (often still is) poor quality
  - Aim: add process, methods, and ensure quality
- **All SE Processes are designed to improve software quality**
- The **Software Quality Team** aim to make sure:
  - A good SE Process is chosen for the project
  - All aspects of that SE Process are all ‘done well’

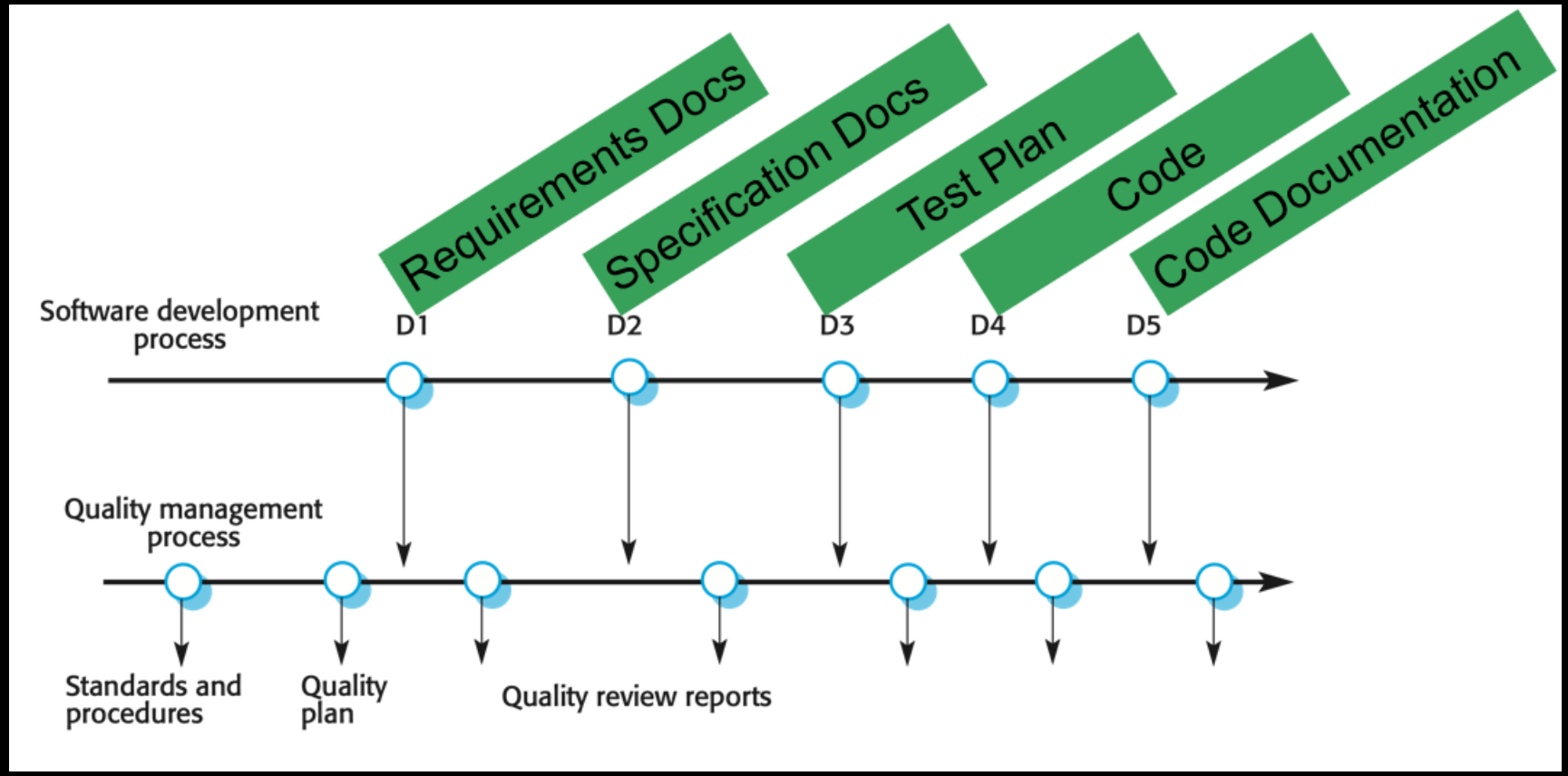
QA Teams - 3 things

1. **Planning** for Quality (to help create Project Plans)
2. **Defining** standards and procedures for the whole company
3. **Checking** that projects conform to company standards
   - So that the company doesn't release products ‘below it’s standard’
- This team is often called the **Quality Assurance** (QA) Team
  - Ideally this team should be separated from dev teams
  - And report to the managers above the project manager

QA Team - Project Activities

"The QA process checks the project deliverables to ensure that they are consistent with organisational standards and goals."

QA Team - Project Activities



QA: Three (four) Things

1. Planning for quality
2. Defining standards
3. Checking for quality
   a) Identifying points for inspections
   b) Identifying relevant measures

QA: Three (four) Things

1. **Planning for quality**
2. Defining standards
3. Checking for quality
   a) Identifying points for inspections
   b) Identifying relevant measures

Planning for Quality

- A Project Quality Plan - sets the bar for the team to achieve
  - It means everyone knows what will be acceptable at the end
  - **"It therefore defines what 'high-quality' software actually means for a particular project"**
- Key Sections of a Quality Plan Document for a Project
  - Product introduction - intended market, etc
  - Product plans - critical release dates, maintenance, etc
  - Process description - what process will be followed for this project
  - **Quality goals** - critical quality attributes for final product
  - Risks & Risk Management - expected key risk areas.

SE Process

- A software engineering process is the **model** chosen for managing the creation of software from initial customer inception to the release of the finished product
- Usually involve techniques such as:
  - Analysis
  - Design
  - Coding
  - Testing
  - Maintenance

Planning for Quality

- Identify standards that will be met
- Recommend **quality-promoting processes** for the project
- Identify what will count as good quality for that project
- So that everyone knows what the **target quality** is
  - (Rather than the target functionality (from Reqs/Specs))
- A QA Team will learn from project to project

Planning for Quality

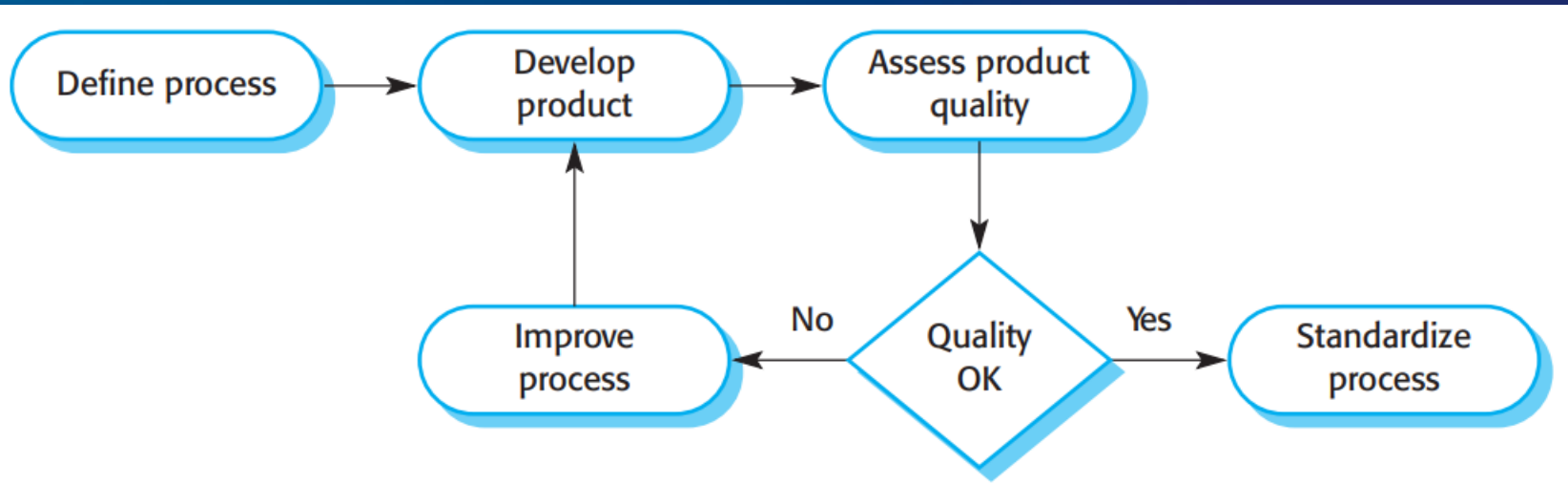


Figure 24.3 Process-based quality

- Aim of the QA Team:
  - To improve the process between projects
  - To document what worked well, and apply successful processes to other projects
- **Aim: to improve the QA standards in the company**

QA: Three (four) Things

1. Planning for quality
2. **Defining standards**
3. Checking for quality
   a) Identifying points for inspections
   b) Identifying relevant measures

Using Standards

- Product Standards - e.g. **documentation standards**, **coding conventions**, **class structure**, etc
- Process Standards - e.g. when reviews / testing etc is done
  - Defined **good practices** at each stage of the SE process

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

ISO Standards exist to help

- ISO 9000 standards on Quality Management Processes
  - It's a bit meta - how to do quality standards
- ISO 9001 - explains how to make your company compliant with standards
- ISO/IEC 12207 - Software life cycle processes
- ISO/IEC 15504 - Software Process Improvement & Capability Determination
- ISO 9241 - accessibility / human factors standards
- IEC 61508 - standards for safe & secure systems
- For various projects, depending on the context, you might need specific standards
  - Medical software standards, government software standards, security etc

Example: Launch Approval

- Process Standard

2.9. Launch approval

The launch of any user-visible change or significant design change requires approvals from a number of people outside of the core engineering team that implements the change. In particular approvals (often subject to detailed review) are required to ensure that code complies with legal requirements, privacy requirements, security requirements, reliability requirements (e.g. having appropriate automatic monitoring to detect server outages and automatically notify the appropriate engineers), business requirements, and so forth.

Quality-Driven Culture

- Most people think that QA is about prescribing standards
- **“In practice, however, there is much more to quality management than standards and the associated bureaucracy to ensure that these have been followed”**
- The risk is that the Quality Assurance Team become the nasty people that make you have to do unnecessary work
- It’s good to get teams to be in charge of what excellence is for them - and the QA team facilitates / holds them to it

Facilitating Quality Assurance

The launch process is also designed to ensure that appropriate people within the company are notified whenever any significant new product or feature launches.

Google has an internal launch approval tool that is used to track the required reviews and approvals and ensure compliance with the defined launch processes for each product. This tool is easily customizable, so that different products or product areas can have different sets of required reviews and approvals.

- This is about creating effective ways to help teams be fast
  - Not slow them down
- A launch approval tool fits nicely with process per product

QA: Three (four) Things

1. Planning for quality
2. Defining standards
3. **Checking for quality**
   a) **Identifying points for inspections**
   b) Identifying relevant measures

QA Reviews & Inspections

- Quality Reviews - and Reports
  - Reviews/inspections of the quality at each stage

QA Reviews & Inspections

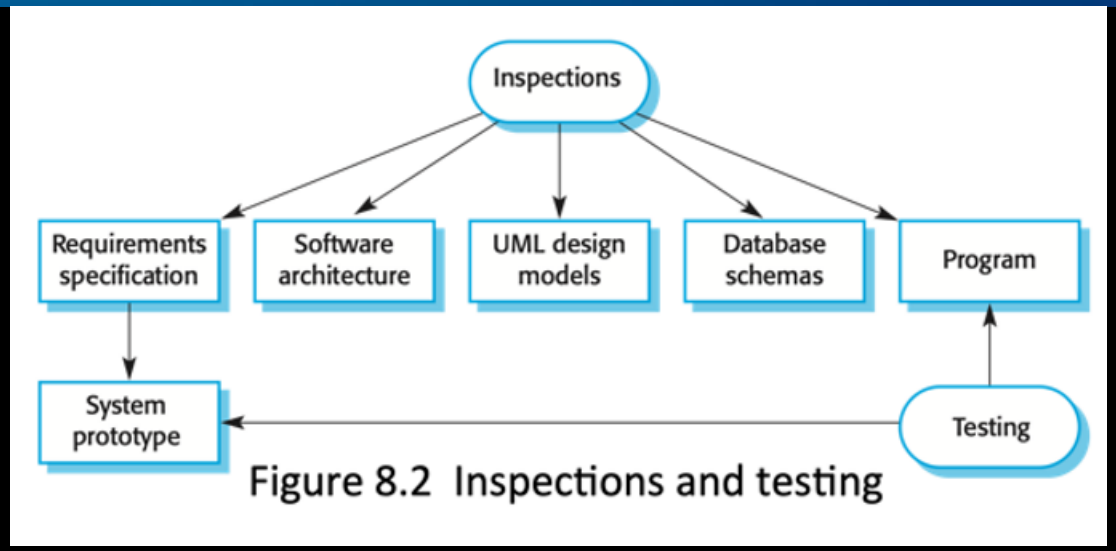


Figure 8.2 Inspections and testing

- We already discussed inspections during coding process
- Paired coding does informal coding inspections
- Fagan (1986) estimated that 60%+ of bugs can be found by inspecting code
- Prowell (1999) claims they found more than 90% of defects

Traditional Inspections

- A group of 3-7 people examine a concrete SE output
  - specific reviewers from QA team
  - project manager, senior designer,
  - 1 or 2 persons who led/built the thing being reviewed
- About 2 hours
- Focus on
  - finding problems and non-conformance to standards
  - checking for completeness
  - or missing or incorrect logical steps
  - do code segments, diagrams, tests, reqs, specs make sense?
- Produce documentation - evidence of quality for client acceptance

Inspection Roles

- Review Leader
  - chairs the meeting
- Recorder
  - keeps track of issues - the documentation
- Reader
  - person who reads through the design or code
- 1 or more Reviewers
  - identify the problems

Fagan Inspections (IBM, 1986)

- A special form of Formal Inspections
  - specialised for **inspecting code**
- More Specific Roles:
  - Moderator (Review leader)
  - Designer - person who designed the code
  - Coder/Implementer - person responsible for code
  - Tester - person who has/will test it

QA Reviews & Inspections

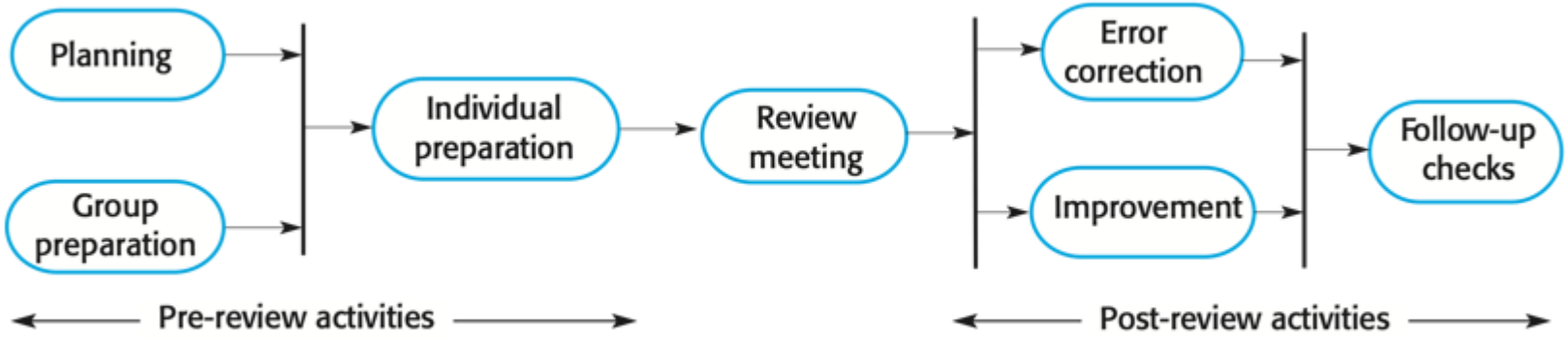


Figure 24.7 The software review process

- Ideally people independently examine the doc in advance
  - and are familiar with it
- The review should take place - a person reads through the document/code
  - focusing on **finding**, **not solving**, problems
- Output is a list of problems that need fixing
  - and should then be fixed - and fixes audited for completion

QA Reviews & Inspections

Fault class	Inspection check
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?
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 deallocated after it is no longer required?
Exception management faults	• Have all possible error conditions been taken into account?

QA Reviews & Inspections

ID	Description	Location(e.g. line num)	Importance	Severity	Assign to
1	Insufficient commenting in Main Class	Main class - line 150	4	2	Tech Lead
2					
3					
4					

Common Inspection Problems

- Criticising the person who built the document
  - not discussing how to make the document better
  - the moderator should make sure this doesn't happen
- People worry that their performance will be judged
  - e.g. by managers or employers
  - could avoid inviting people's line managers to the inspection
- People don't properly prepare before the inspection
  - make sure people have carefully read the document first
- People try to discuss every problem as they find it
  - and very little gets actually inspected

Agile Inspections

- Traditional inspections are important, and common
  - but Paired Programming is more agile inspection for coding

> **2.3. Code Review**
>
> **Google has built excellent web-based code review tools**, integrated with email, that allow authors to request a review, and allows reviewers to view side-by-side diffs (with nice color coding) and comment on them. When the author of a change initiates a code review, the reviewers are notified by e-mail, with a link to the web review tool's page for that change. Email notifications are sent when reviewers submit their review comments. In addition, automated tools can send notifications, containing for example the results of automated tests or the findings of static analysis tools.
>
> **All changes to the main source code repository MUST be reviewed by at least one other engineer.** In addition, if the author of a change is not one of the owners of the files being modified, then at least one of the owners must review and approve the change.

- But this isn't practical for e.g. Design Reviews, Test Plan Reviews
  - or Requirements Review, with customer

Retrospectives vs Postmortems

- Both are a form of review/inspection
- Agile Retrospectives
  - e.g. **after each sprint** (or each incremental delivery)
  - Dev team reflects on progress, good and bad things that happened, etc
  - Can we optimise anything? Avoid anything? Try anything?
  - self-driven by dev team
  - FEEDS INTO: next round of dev on **same** project
- Agile Postmortems
  - **End of project review** - focus on issues experienced
  - Considered hard to do without blaming/being negative
  - More likely to be led by QA team, with Devs
  - FEEDS INTO: QA process improvement for **future** projects

Measuring Quality

- There are many things we might try to ask/answer:
  - » have code and documentation standards been followed?
  - » has the software been properly tested?
  - » is the software sufficiently dependable/reliable?
  - » is the performance acceptable?
  - » is the software usable?
  - » is the software well structured and understandable?
- The QA Team help to define how to answer these questions
  - in a way that is corporately acceptable/agreeable

Measuring Quality

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

- These non-functional aspects that often define quality
- These are hard to measure, and sometimes conflict
- Part of a Quality Plan is to set the Quality Goals
  - this is to consider which are most important

Things you *can* measure

- Some of these subjective measures of quality
  - can be indirectly inferred from objective metrics
- These are things that can be actually counted - e.g.:
  - » number of lines of code
  - » the Fog Index (readability of comments)
  - » number of reported faults reported after delivery
  - » number of person days for coding

Measure Types

- Code/Predictor Measures
  - help you to judge aspects of code quality
  - but they are only predictors of quality
    e.g. predictors of maintainability
    e.g. predictors of substandard quality
- Control/Process Measures
  - measuring the success of your SE Process
  - help decide whether to improve processes later
  - we see this more in the Project Management lectures

Things you *can* measure

- Some of these subjective measures of quality
  - can be indirectly inferred from objective metrics
- These are things that can be actually counted - e.g.:
  - » number of lines of code
  - » the Fog Index (readability of comments)
  - » number of reported faults reported after delivery
  - » number of person days for coding

Code/Predictor Measures

Control/Process Measures

Measures - but of what?

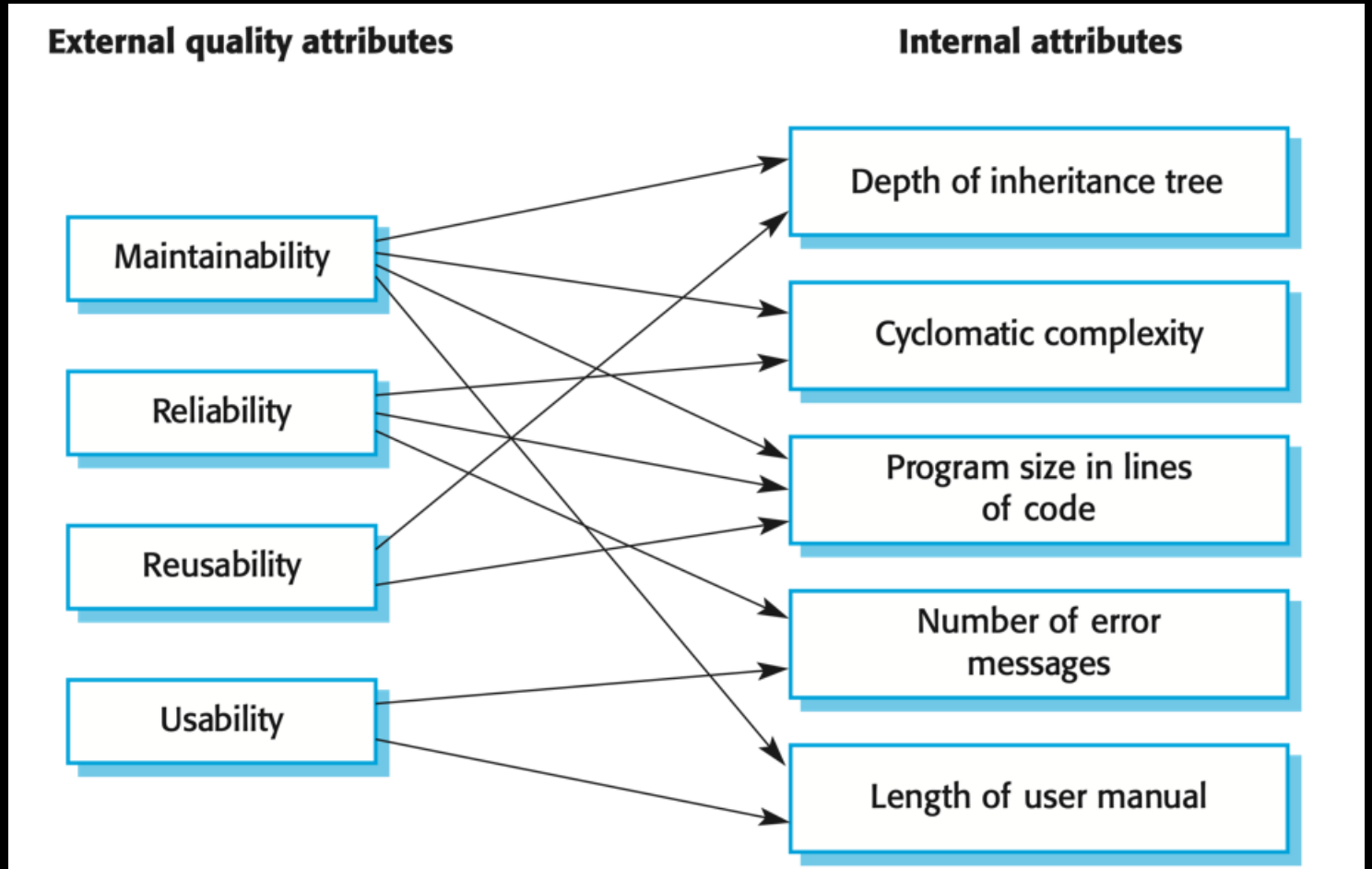


Figure 24.10 Relationships between internal and external software

Measures - but of what?

Software metric	Description
Fan-in/Fan-out	Fan-in is a measure of the number of functions or methods that call another function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.
Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss cyclomatic complexity in Chapter 8.
Length of identifiers	This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if-statements are hard to understand and potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value of a document's Fog index, the more difficult the document is to understand.

Measures - but of what?

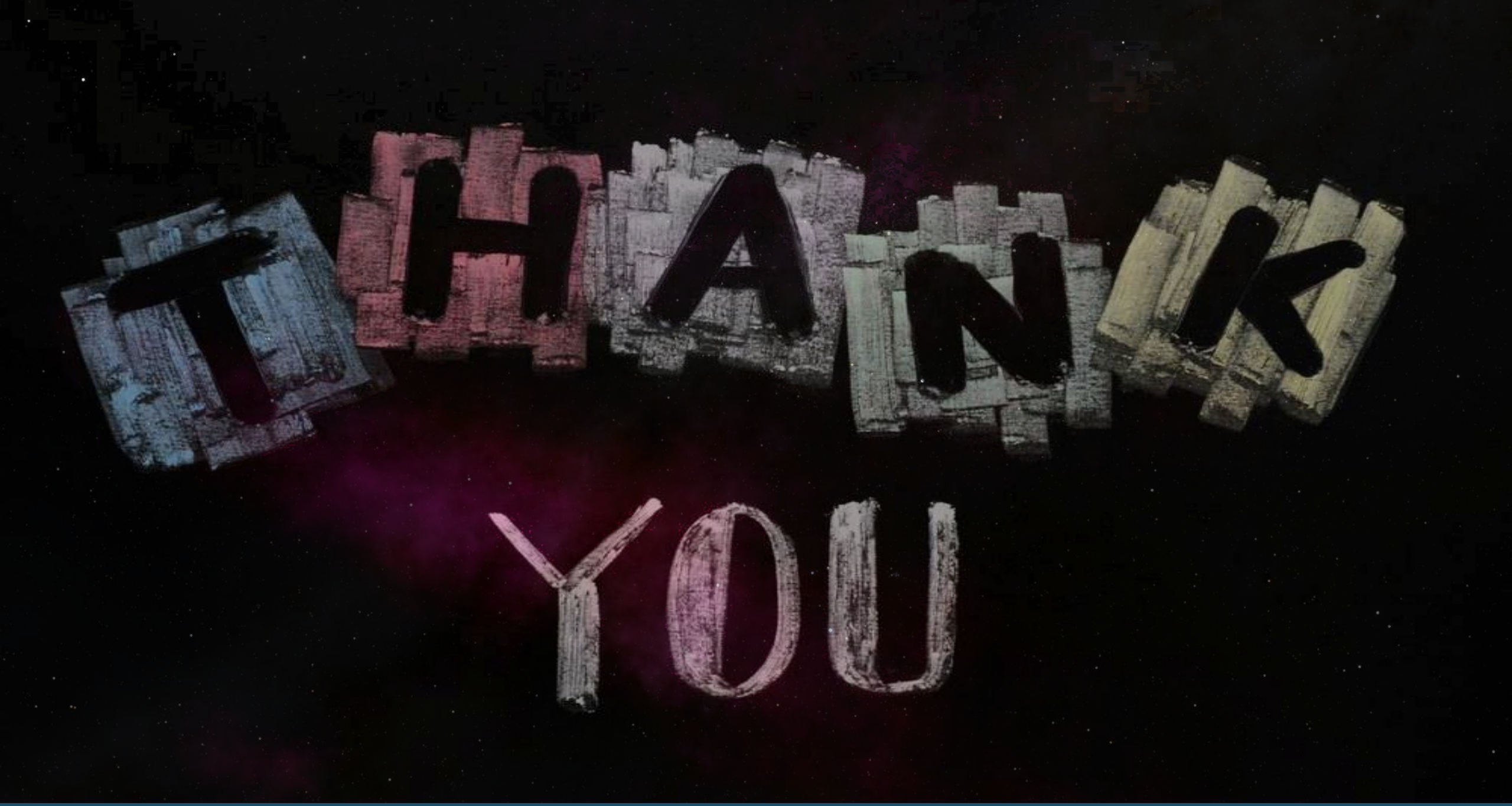
Object-oriented metric	Description
Weighted methods per class (WMC)	This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand. They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
Depth of inheritance tree (DIT)	This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design. Many object classes may have to be understood to understand the object classes at the leaves of the tree.
Number of children (NOC)	This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.
Coupling between object classes (CBO)	Classes are coupled when methods in one class use methods or instance variables defined in a different class. CBO is a measure of how much coupling exists. A high value for CBO means that classes are highly dependent, and therefore it is more likely that changing one class will affect other classes in the program.
Response for a class (RFC)	RFC is a measure of the number of methods that could potentially be executed in response to a message received by an object of that class. Again, RFC is related to complexity. The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors.
Lack of cohesion in methods (LCOM)	LCOM is calculated by considering pairs of methods in a class. LCOM is the difference between the number of method pairs without shared attributes and the number of method pairs with shared attributes. The value of this metric has been widely debated and it exists in several variations. It is not clear if it really adds any additional, useful information over and above that provided by other metrics.

Measures - but of what?

- We can only make limited assumptions about code measures
- What a metric predicts can depend on the language used
  - so its hard to compare across projects/languages
- Its still relatively 'to be proven' whether each of these metrics are 'good'
- Metrics can **only be used as predictors**
  - once they have been correlated with past project success
  - companies develop metrics over time that work for them

Summary

- Quality Assurance is a lot more than release/acceptance testing
- Quality Assurance - is about planning for high quality goals
- Quality Assurance - is about everyone aiming for high quality
- Quality Assurance - is about inspecting for quality at each stage
- Quality Assurance - is about taking / learning from metrics
  - of software quality - and process quality


THANK
YOU