SOFTWARE ENGINEERING
LECTURE 7

CARSTEN RUSENG JAKOBSEN

AALBORG UNIVERSITY
DENMARK

Some reflections from review of miniprojects

- Use table from Böhm and Turner to describe
- Use diagram to for process description. You probably have combined elements from e.g. waterfall and iterative – a diagram will show more clearly how you have blended the elements
- Plandriven / agile is not black/white – you mostly have a mix
- Practices of Scrum and XP – why are they important, how do they differ
- Phases of UP are not equal to waterfall
- Incremential can be plandriven or agile
  - Sommersville conceptually describe incremental and iterative together
  - In the industry incremental is considered plandriven and iterative agile
- Agile is not a process, but values based on a manifesto

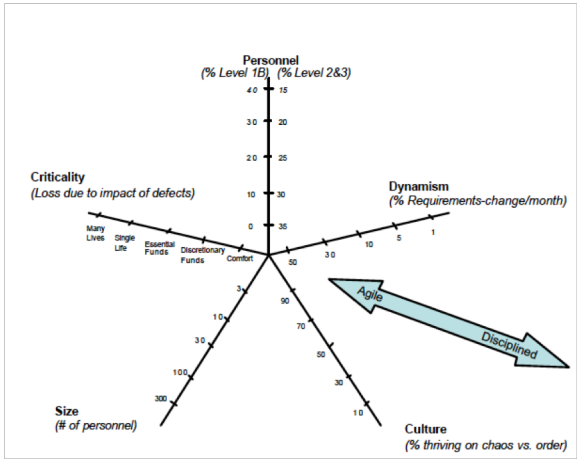
The table from Böhm and Turner

The table provides a structured way to argue for where to place the 5 parameters in home ground

The ideal customer for agile projects are (CRACK):

- **Collaborative:** they will engage with their customer peers and with the development team
- **Representative:** they know the product or system requirements and can represent their constituents accurately
- **Authorized:** they can make the decisions needed to keep velocity up, and their decisions stick!
- **Committed:** they take their job seriously and make every effort to advance project success
- **Knowledgeable:** they can understand what the developers are telling them in the context of the business or market situation

Characteristics	Agile	Plan-driven
Application		
Primary Goals	Rapid value; responding to change	Predictability, stability, high assurance
Size	Smaller teams and projects	Larger teams and projects
Environment	Turbulent; high change; project-focused	Stable; low - change; project/organization focused
Management		
Customer Relations	Dedicated on-site customers; focused on prioritized increments	As-needed customer interactions; focused on contract provisions
Planning and Control	Internalized plans; qualitative control	Documented plans, quantitative control
Communications	Tacit interpersonal knowledge	Explicit documented knowledge
Technical		
Requirements	Prioritized informal stories and test cases; undergoing unforseeable change	Formalized project, capability, interface, quality, forseeable evolution requirements
Development	Simple design; short increments; refactoring assumed inexpensive	Extensive design; longer increments; refactoring assumed expensive
Test	Executable test cases define requirements, testing	Documented test plans and procedures
Personnel		
Customers	Dedicated, collocated CRACK* performers	CRACK* performers, not always collocated
Developers	At least 30 percent full-time Cockburn Level 2 and 3 experts; no Level 1B or -1 personnel**	50 percent Cockburn Level 3s early; 10 percent throughout; 30 percent Level 1Bs workable; no Level -1s**
Culture	Comfort and empowerment via many degrees of freedom (thriving on chaos)	Comfort and empowerment via framework of policies and procedures (thriving on order)
* Collaborative, Representative, Authorized, Committed, Knowledgable ** See Table 2. These numbers will particularly vary with the complexity of the application		

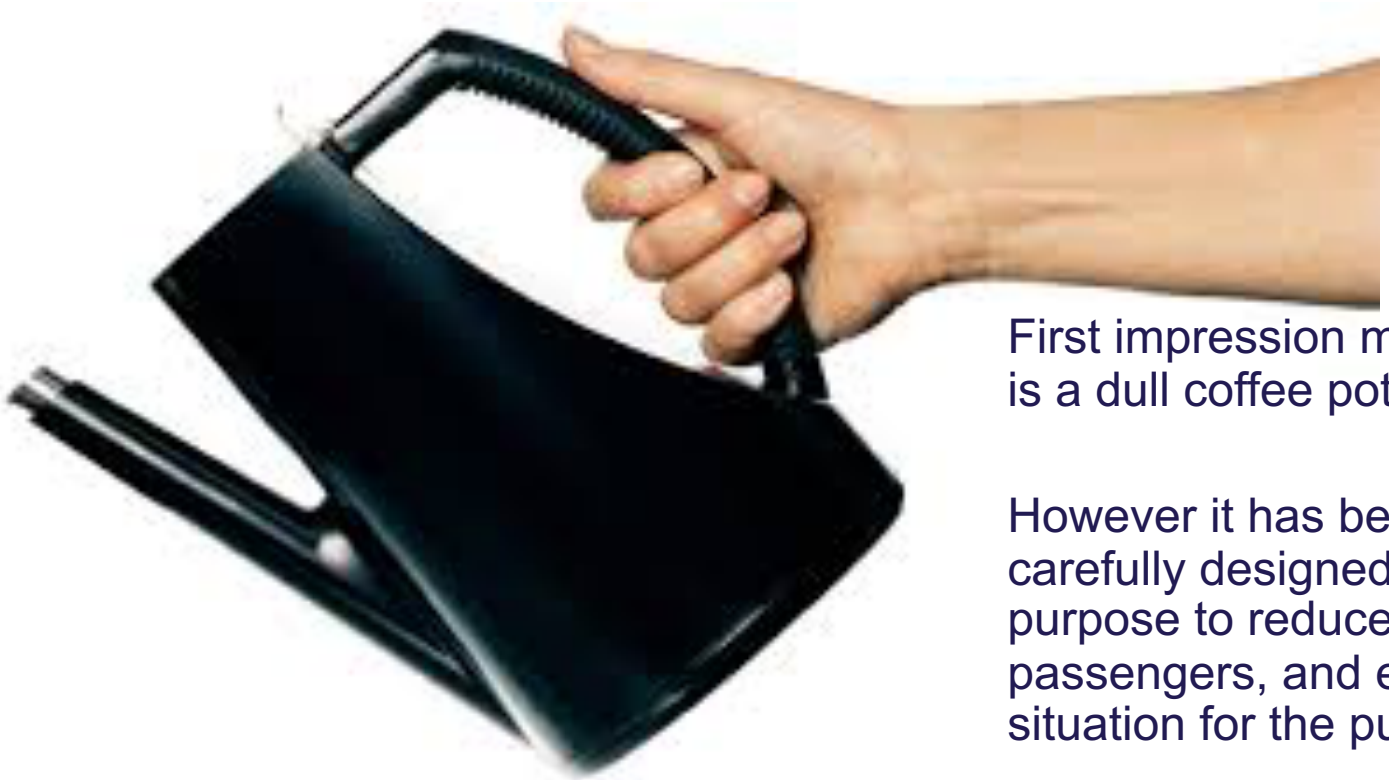


Home ground

What is quality?

- What is a good program?
- What is a good user interface?
- What is a good requirements specification?
- What is a good project plan?

Quality may not be obvious – Coffe pot for flights



First impression may be: this is a dull coffee pot.

However it has been carefully designed for it´s purpose to reduce risk for passengers, and ergonomic situation for the purser.

- No spill, deep gravity, minimal movement to pour coffee

Definition of quality

- **Quality** is a reflection of one or more peoples' assessment of correspondance between their expectations and experience of a product or a service

- Quality can be divided into categories
  - Product quality
  - Proces quality
  - Quality of expectations

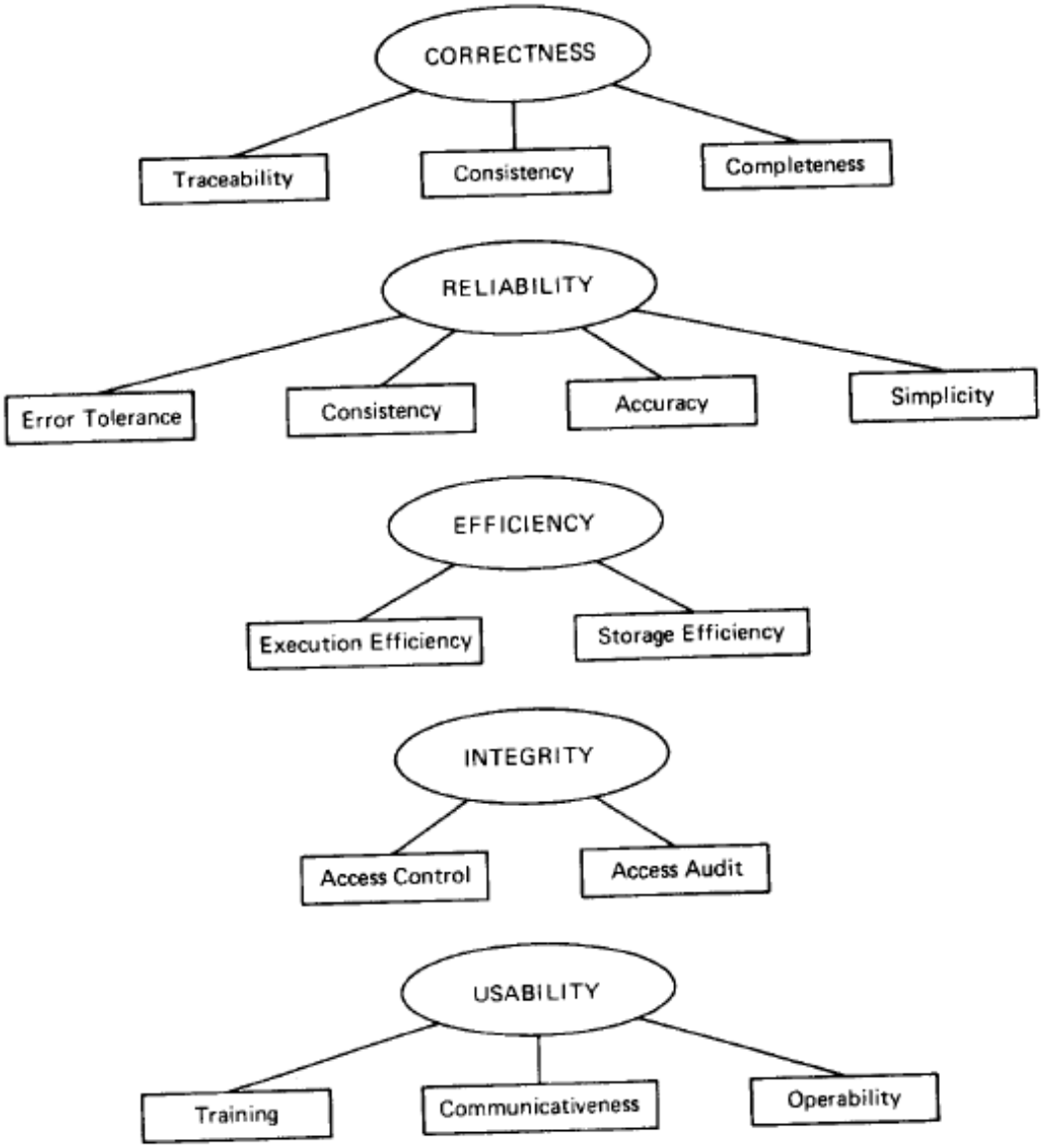
- Different categories of quality are measured with different measures

Software quality attributes

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

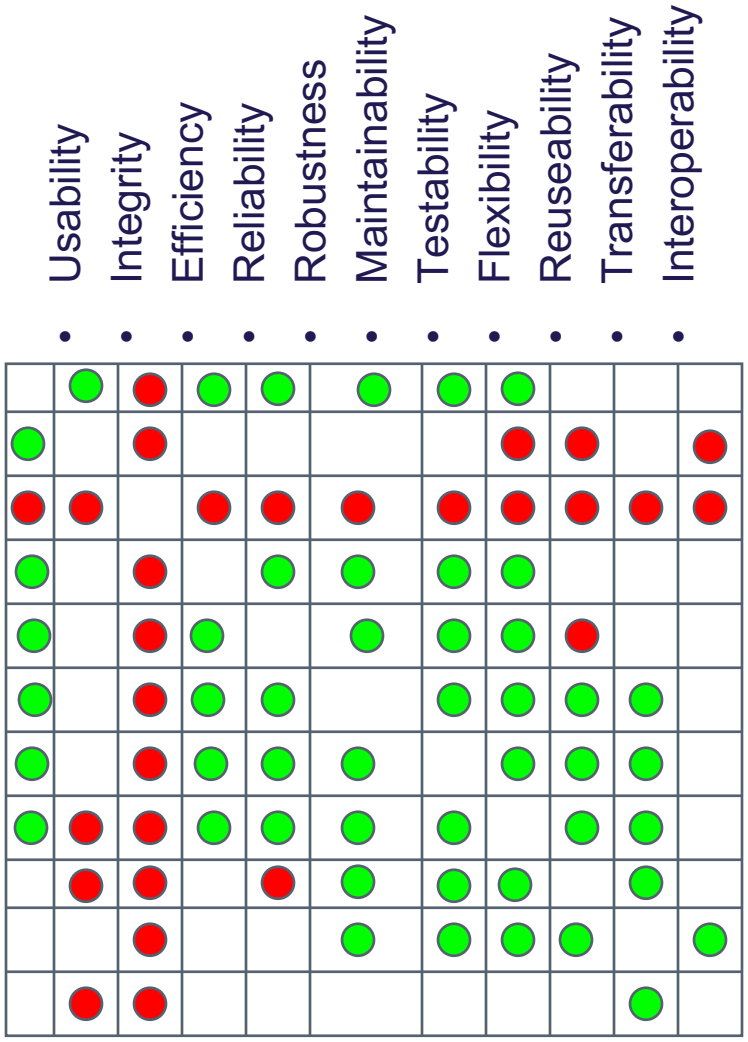
(Sommerville)

SW quality factors



(McCall et al.)

Quality Trade-offs – you can't have all



Positive influence
Negative influence

- Usability
- Integrity
- Efficiency
- Correctness
- Reliability
- Maintainability
- Testability
- Flexibility
- Reuseability
- Transferability
- Interoperability

(OOA&D)

Cost of bad quality (fixing errors)

- Direct cost of error correction
  - Loss
  - Wasted work
  - Maintenance usually have larger costs than development
- Indirect cost of error correction
  - Follows from poor quality
  - Has potentially severe consequences

- Quality Management reduces these costs significantly

Quality Management

- Quality Management consists of
  - Quality Assurance (Plan or design processes to prevent bad quality)
  - Quality Control (Monitor that work products meet quality standards)

” Software Quality Management techniques have their roots in methods and techniques that were developed in manufacturing industries, where the terms *quality assurance* and *quality control* are widely used.

Quality assurance is the definition of processes and standards that should lead to high quality products and the introduction of quality processes in the manufacturing process.

Quality control is the application of these quality processes to weed out products that are not of the required level of quality. Both quality assurance and quality control are part of quality management.

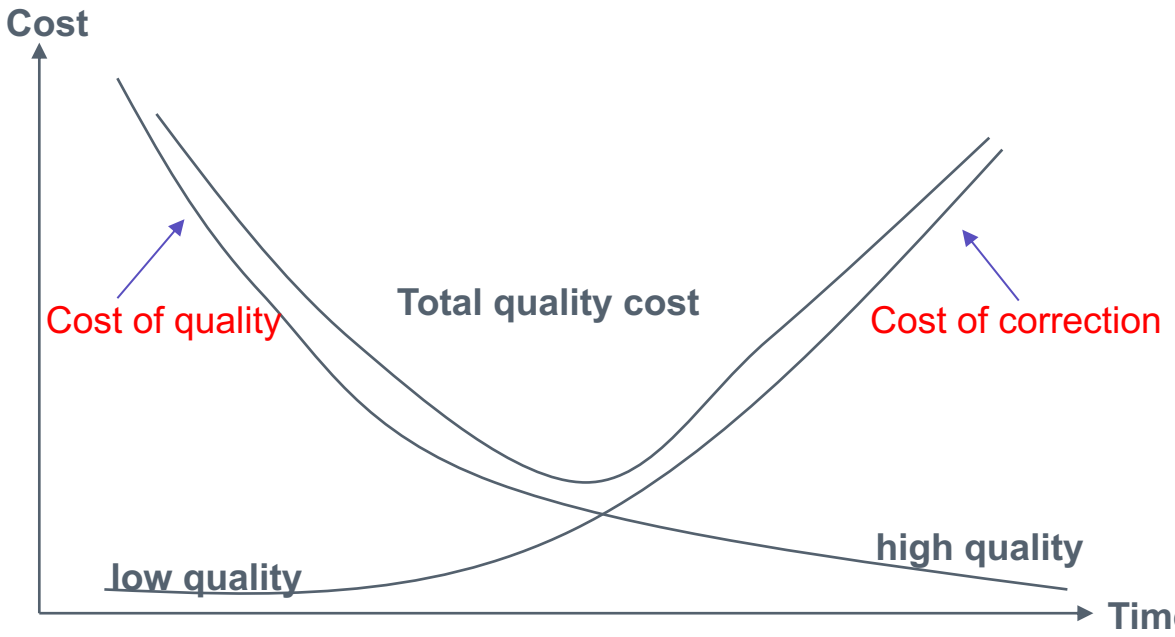
In the software industry, some companies see quality assurance as the definition of procedures, processes, and standards to ensure that software quality is achieved. In other companies, quality assurance also includes all configuration management, verification, and validation activities that are applied after a product has been handed over by a development team”

- Sommerville

Cost of quality management

- Preventive costs
  - Better planning
  - Measurable quality factors
  - Education and training of developers
- Monitoring costs
  - Evaluation, peer review, code inspection
  - Testing

Overall theory of cost and quality



- Low quality (low quality management) is initially cheap, but becomes gradually very expensive
- High quality management, has an initial cost when quality processes are defined, but is cheaper later because users are reporting much less errors, and code is more stable
- The amount of quality management should be balanced to the cost – a process that is 100% defect free is often too expensive, so an appropriate compromise is normally made. As a rsult the initial cost is a little less than for very high quality management, at the cost that slightly more defects are reported over time.
- We plan and design, how and when to do *verification* and *validation* in our process

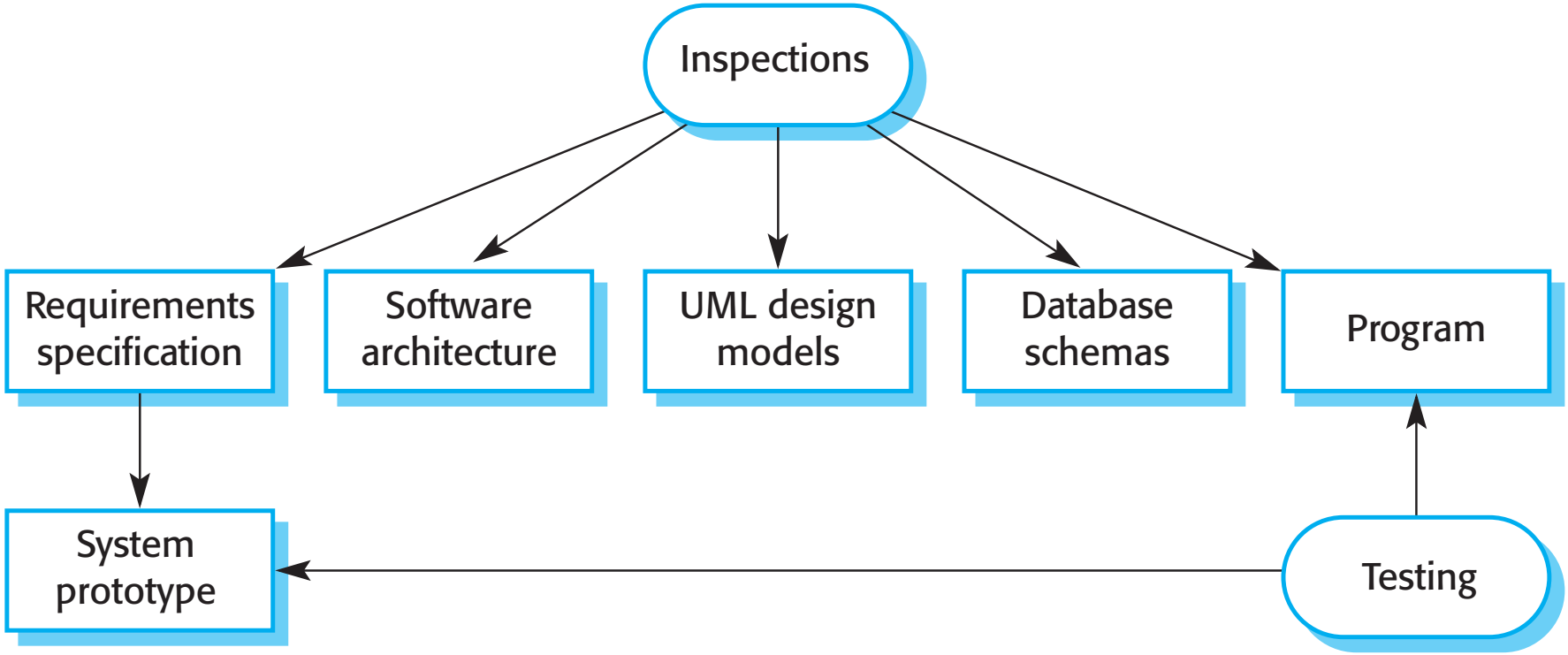
Quality Assurance

- Validation: (fit for use)
  - Are we building the right systems?
  - Conforms to customers' expectations and experience
- Verification: (Are all requirements implemented)
  - Are we building the system right?
  - Conforms to specification
- Techniques:
  - Testing of programs and prototypes
  - Reviewing of specifications, documentation and programs

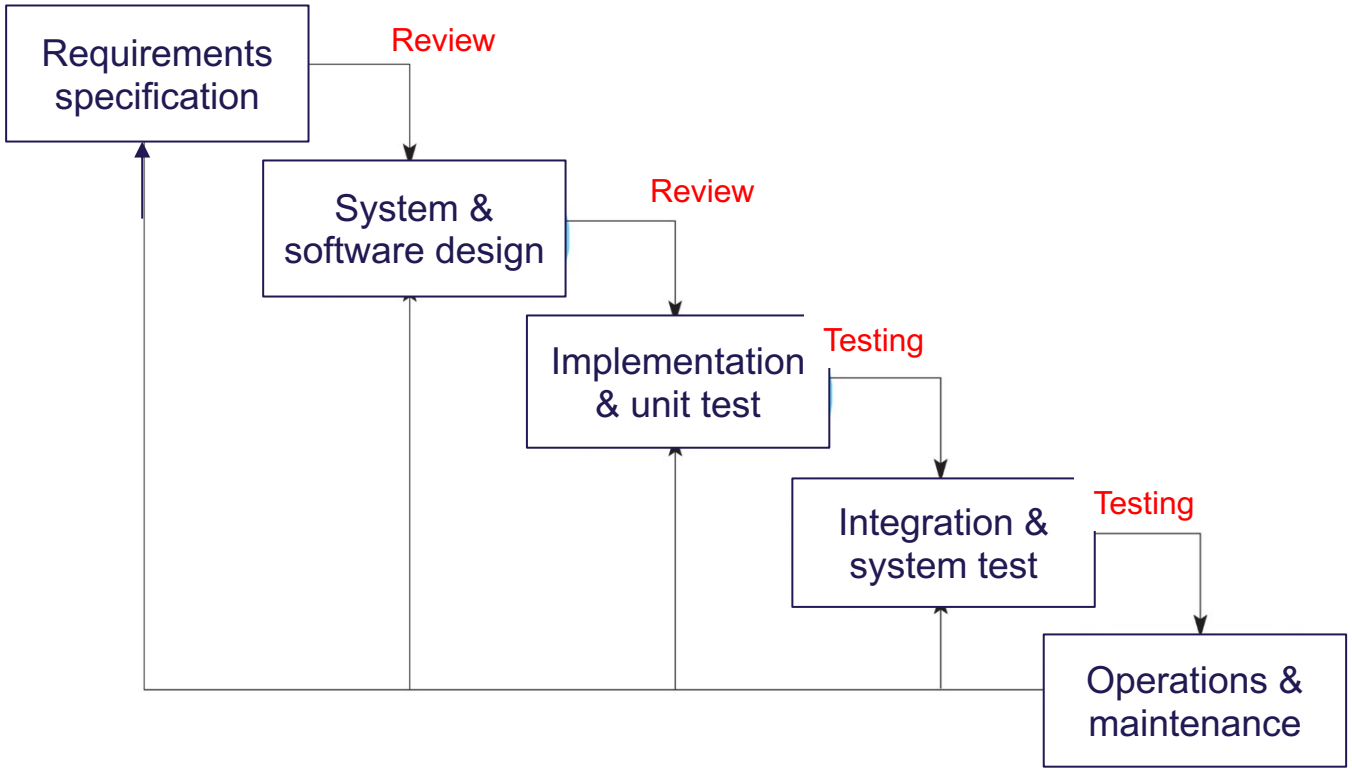
Inspections (reviews) and testing

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

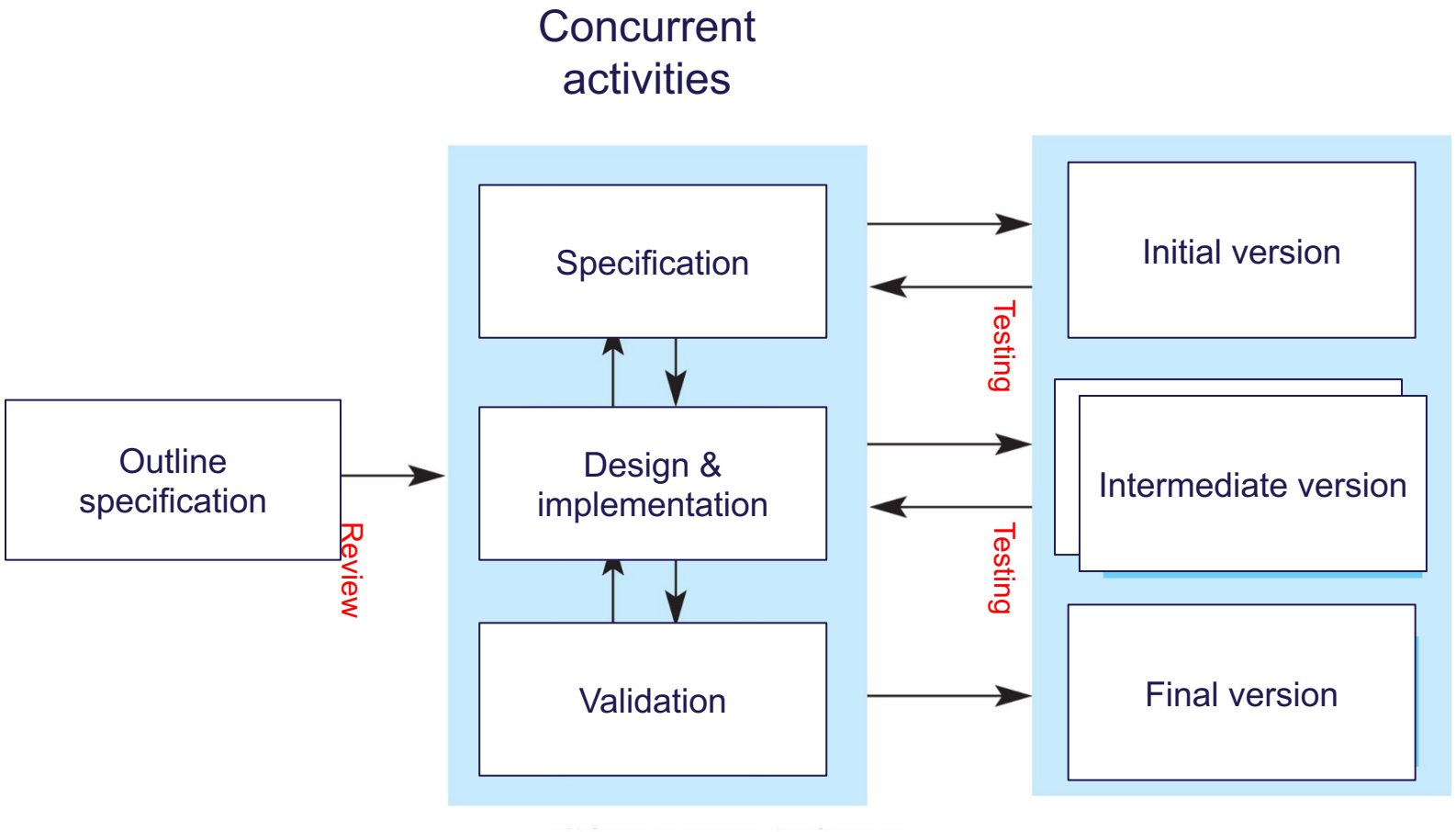
Inspections and testing



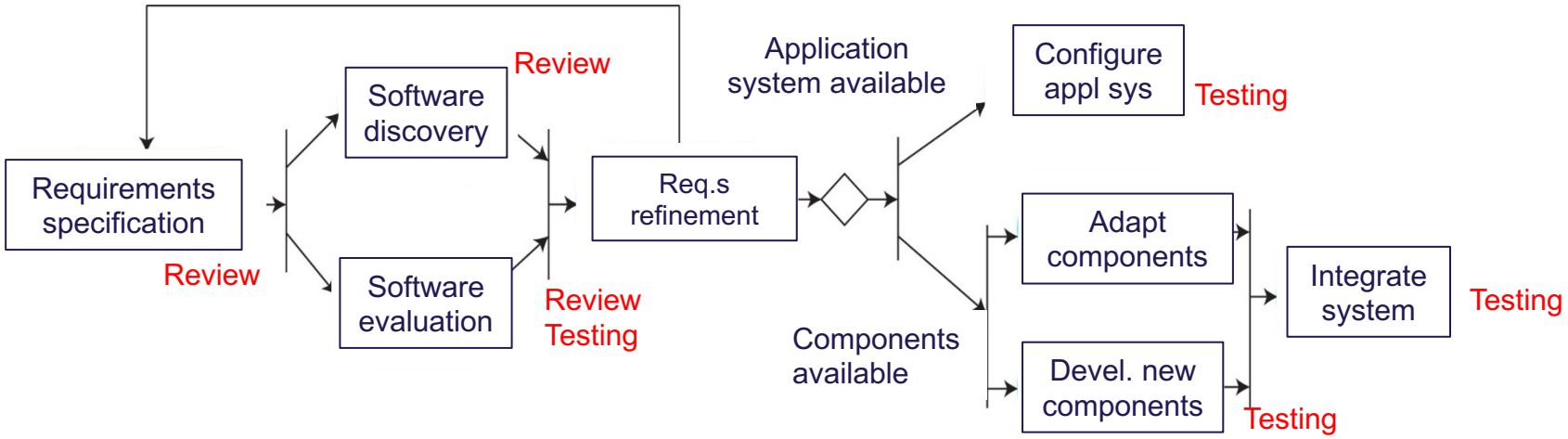
Waterfall Model



Incremental Model



Integration & configuration



Quality management and agile development

- Quality management in agile development is informal rather than document-based.
- It relies on establishing a quality culture, where all team members feel responsible for software quality and take actions to ensure that quality is maintained.

Quality practices

- *Definition of Done*
  Team agree on what criteria must be met before a task is complete

- *Sprint review*
  PO and other stakeholders validate the sprint delivery meets expectations

- *Check before check-in*
  Developers are responsible for organizing their own code reviews with other team members before the code is checked in to the build system.

- *Never break the build*
  Team members should not check in code that causes the system to fail. Developers have to test their code changes against the whole system and be confident that these work as expected.

- *Fix problems when you see them*
  If a programmer discovers problems or obscurities in code developed by someone else, they can fix these directly rather than referring them back to the original developer.

Fundamental Process Theory

- A software product can only be as good as the process through which it is produced

- It's a quality perspective
- You can only improve the quality of the product if you improve the process
  - Repeating the same process, will create same level of quality
  - Sources of bad quality can be used as input to improve the process