# Principles of software testing

# Testing identifies the presence of defects

not possible to demonstrate absence of defects, only their presence

# Exhaustive testing is impossible

**possible only in trivial cases**

**Reduce the number of tests designed/executed to make process economically viable**

**Choosing which tests to design and execute; *risk management***

# Early testing

**test early in the development cycle**

**Costs increase; design -1, coding -10,…**
**final product 1,000**

# Defect clustering

**defects are clustered (aggregated) according to identical criteria (similar error repeated, group of components designed in the same period of time, defective off-the –shelf component)…**

# Pesticide paradox

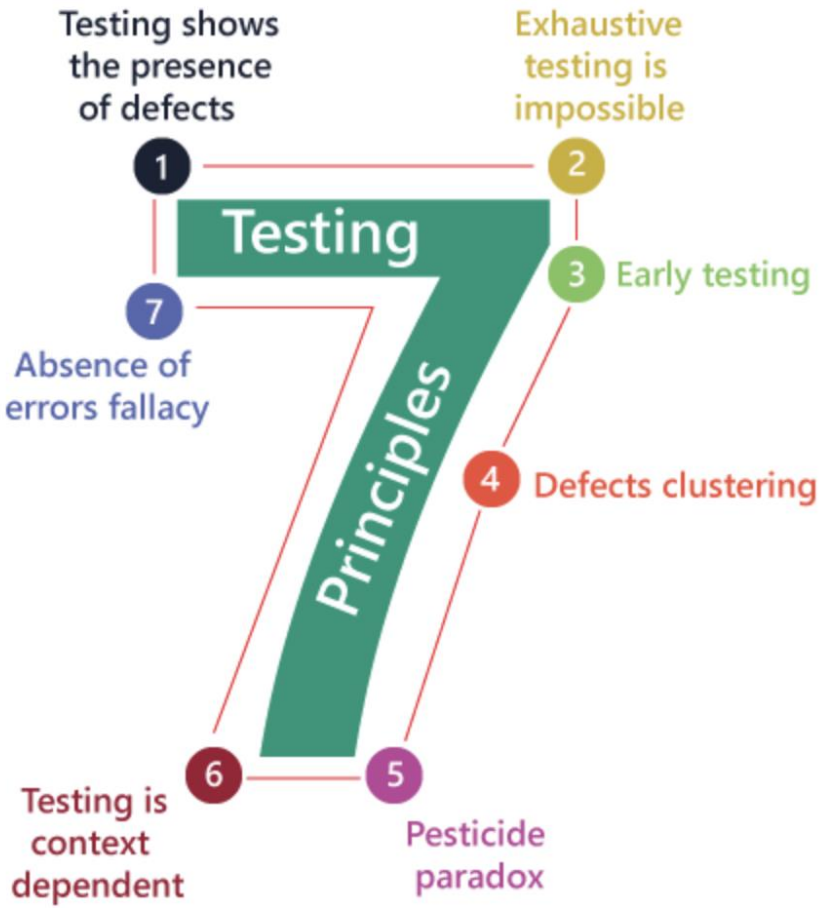
**a lack of efficiency of tests when used over a period of time; re-execution of identical tests**

**Tests are changed over time, variation in test data, order of execution of tests…**

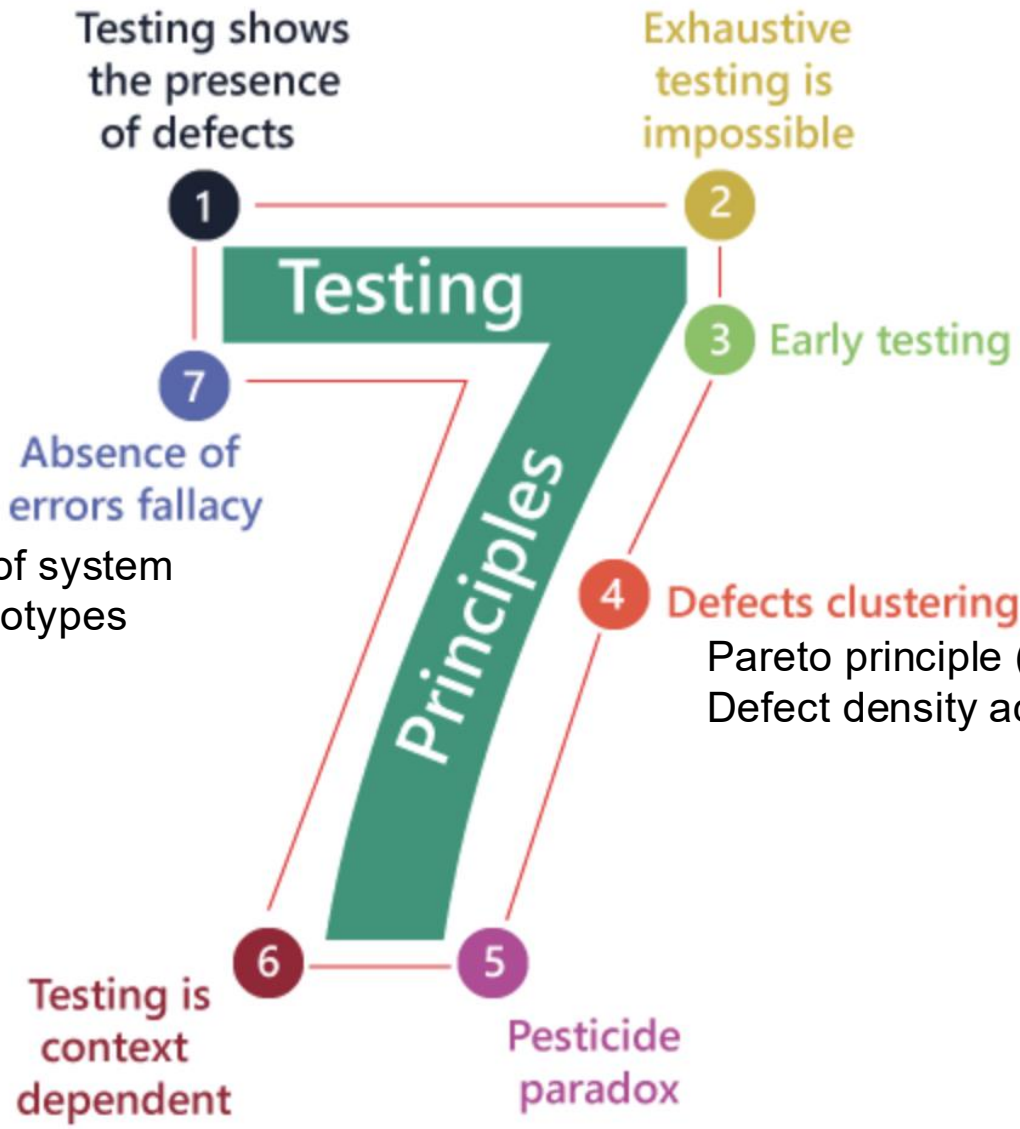
# Testing is context dependent

**safety-critical systems tested differently than most e-commerce systems**

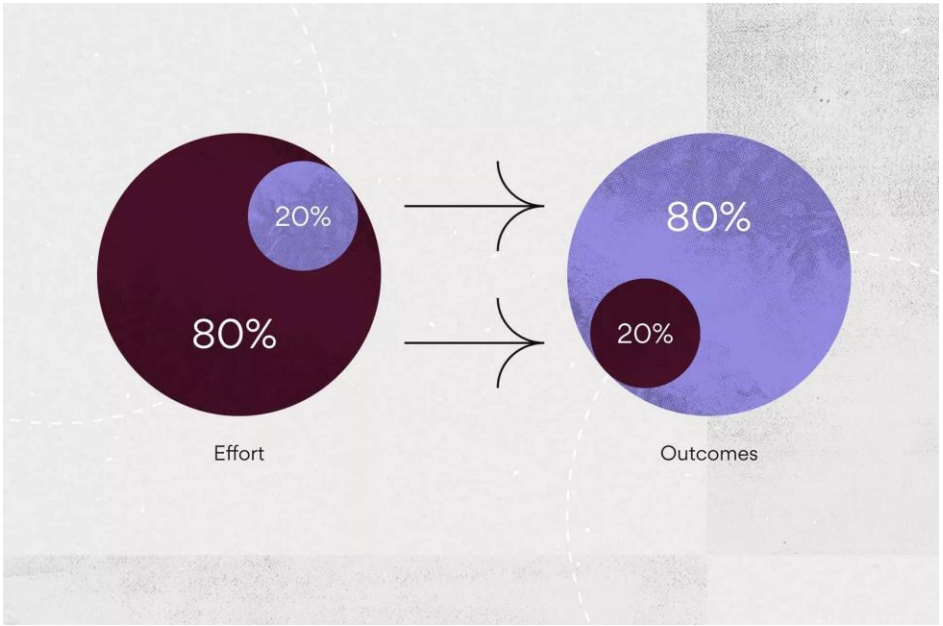
**Test effort influenced by available time and resources**

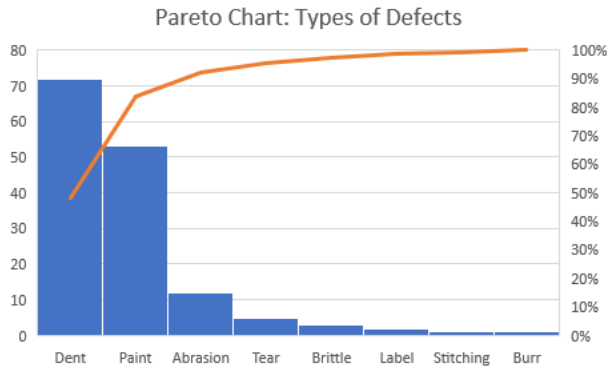

Testing shows the presence of defects
Exhaustive testing is impossible
1
2
Testing
7 Principles
3 Early testing
7
Absence of errors fallacy
4 Defects clustering
6
5
Testing is context dependent
Pesticide paradox

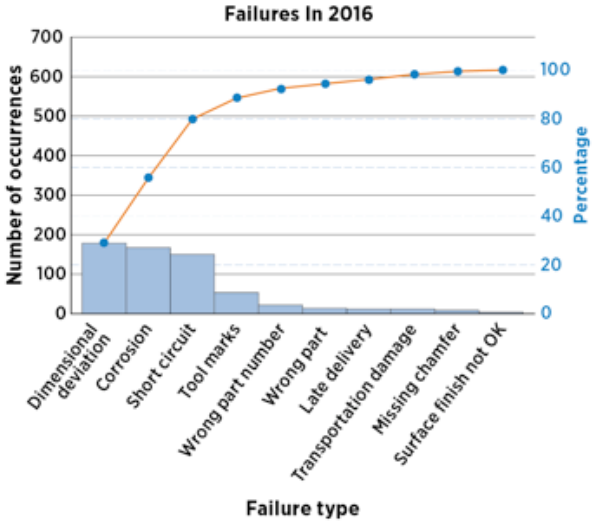

SEVEN again... MAGNIFICENT again!
Hunted by the Law...
Hated by the Lawless!
MIRISCH PRODUCTIONS, INC.
Yul Brynner
in "Return of the Seven"
CO-STARRING
Robert Fuller
AND INTRODUCING
JORDAN CHRISTOPHER
JULIAN MATEOS · WARREN OATES · VIRGILIO TEXEIRA · CLAUDE AKINS · ELISA MONTES ·
COLOR by DeLuxe
PANAVISION
UNITED ARTISTS
alamy stock photo
BP9NNX
www.alamy.com


Testing shows the presence of defects
Exhaustive testing is impossible
1
2
Testing
7 Principles
3 Early testing
7
Absence of errors fallacy
Usability of system
Early prototypes
4 Defects clustering
Pareto principle (80/20)
Defect density across the system
6
5
Testing is context dependent
Pesticide paradox

# Pareto Principle



# Software Quality

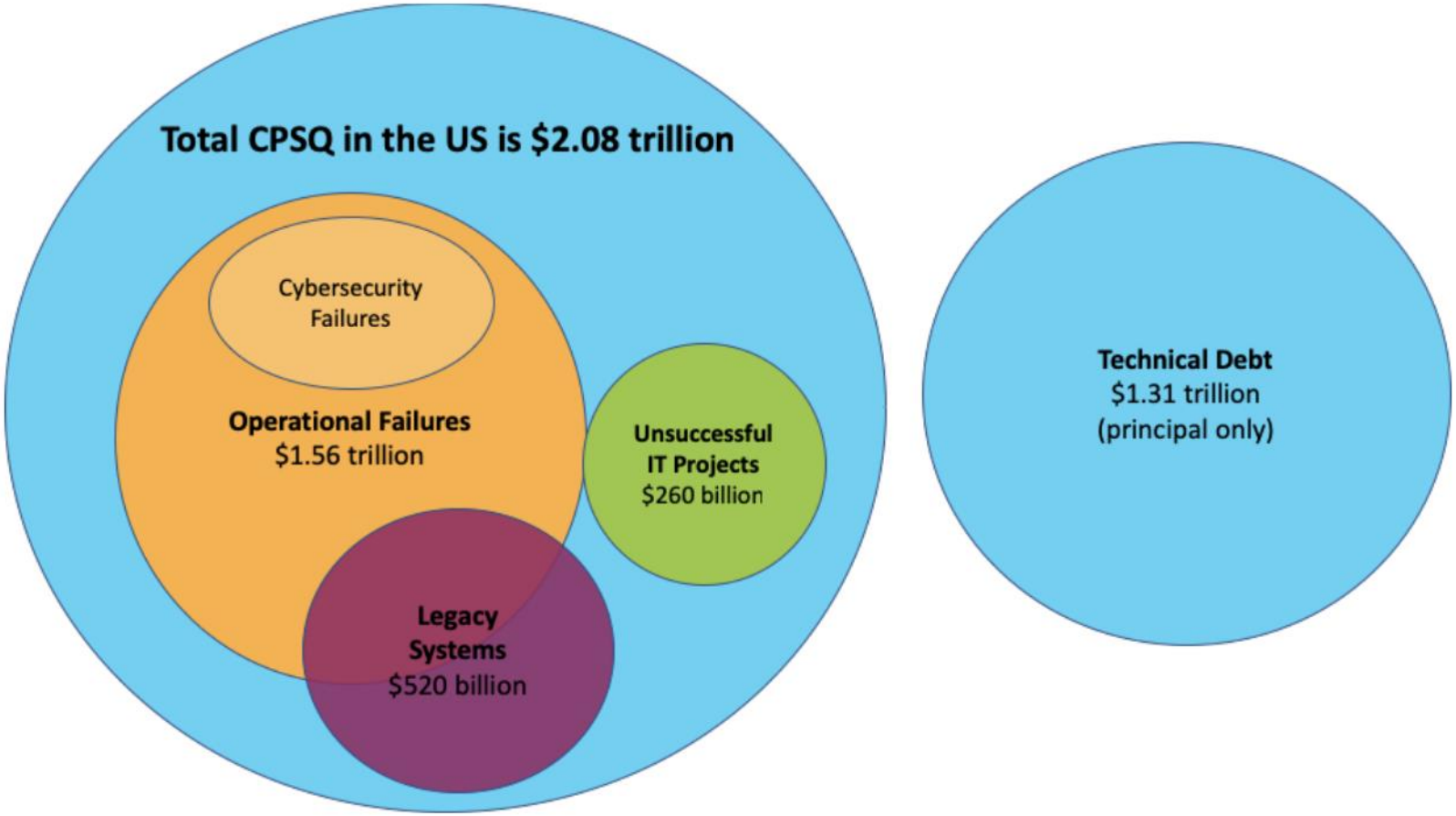
# Software Quality (ISO 9126)

**ISO – International Standardization Organization**
**(quality management system)**

**http://en.wikipedia.org/wiki/ISO_9126**

**hierarchical framework for quality definition**
**(characteristics and sub-characteristics)**

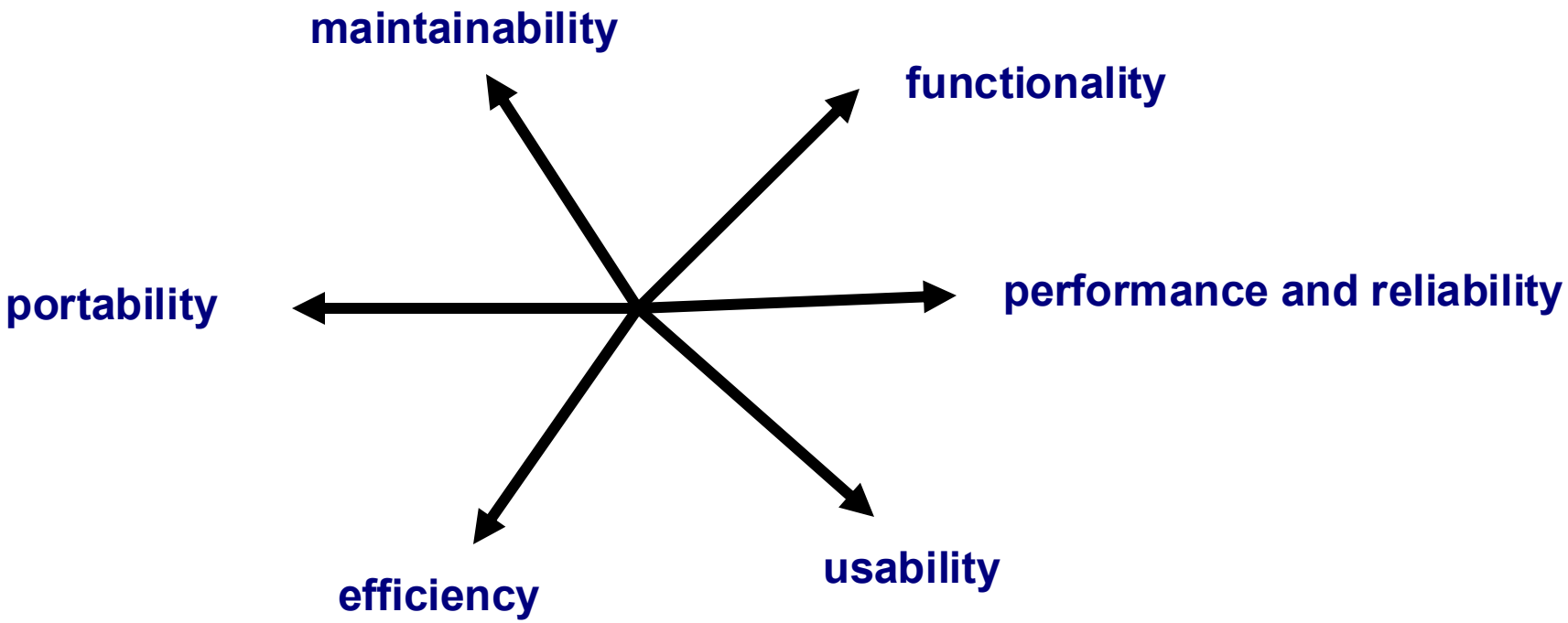
# Cost of Poor Software Quality




The Cost of Poor Software Quality in the US: The 2020 Report

# **Software Quality** (ISO 9126)

***multifaceted* and *hierarchical* concept**



# Software Quality (ISO-9126)

## *Functionality*

***EXISTENCE OF SET OF FUNCTIONS AND THEIR SPECIFIED PROPERTIES***

***suitability***

***accuracy***

***interoperability***

***security***

## *Performance and Reliability*

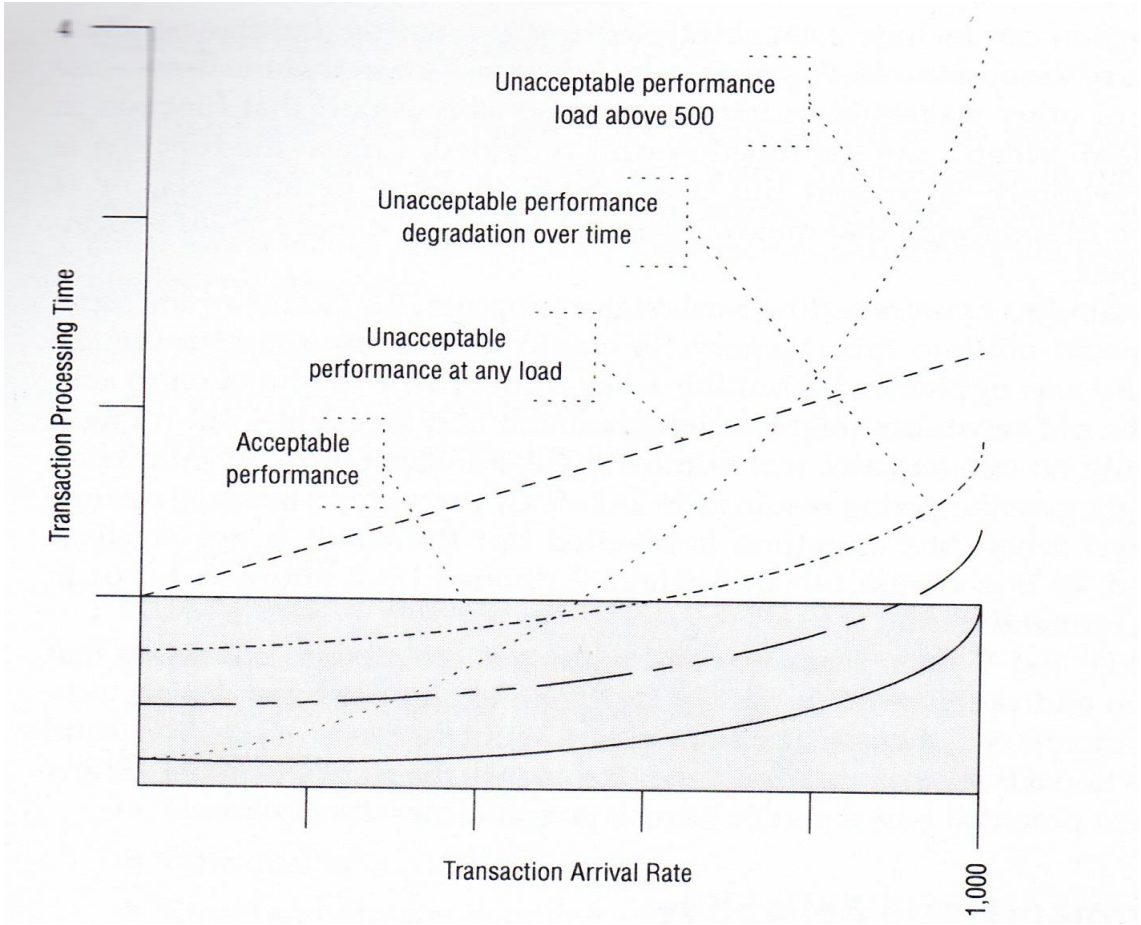
***SOFTWARE MAINTAINS ITS LEVEL OF PERFORMANCE UNDER STATED CONDITIONS AND FOR A STATED PERIOD OF TIME***

***maturity***

***fault tolerance***

***recoverability***

# Software Quality (ISO-9126)



# Software Quality (ISO-9126)

***Efficiency*** — *LEVEL OF PERFORMANCE AND AMOUNT OF RESOURCES USED*

- *time behavior*
- *resource behavior*

***Maintainability*** — *EFFORT NEEDED TO MAKE SPECIFIED MODIFICATIONS*

- *analyzability*
- *changeability*
- *stability*
- *testability*

# Software Quality (ISO-9126)

***Usability*** — *ABILITY OF SOFTWARE CONCERNING THE EFFORT NEEDED FOR USE, ASSESSMENT OF USE (by a stated or implied set of users)*

- *understandability*
- *learnability*
- *operability*

***Portability*** — *ABILITY OF SOFTWARE TO BE TRANSFERRED FROM ONE ENVIRONMENT TO ANOTHER*

- *adaptability*
- *Instability*
- *Conformance*
- *replaceability*

# McCall model for software quality factors

**Product operation factors**

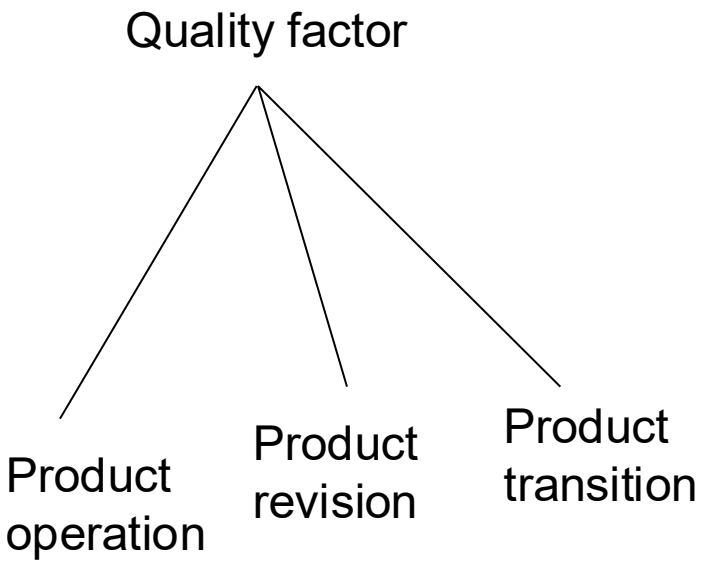
- Correctness
- Reliability
- Efficiency
- Integrity
- Usability

**Product revision factors**

- Maintainability
- Flexibility
- Testability

**Product transition factors**

- Portability
- Reusability
- Interoperability



# McCall model for software quality factors (1)

## Product operation factors

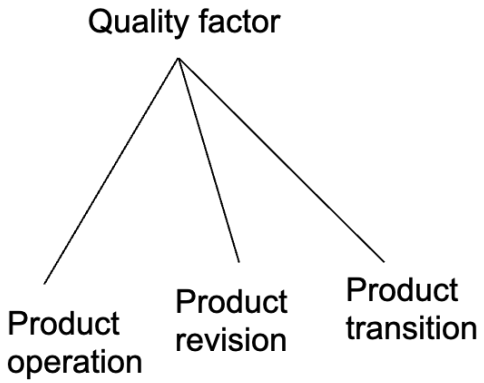
**Correctness**: required accuracy of the output, completeness of output, Up-to-dateness, response time

**Reliability**: deals with failure to provide service, failure rate, system downtown (%), maximum recovery time

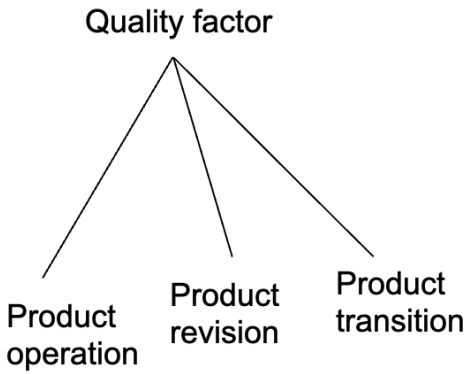
**Efficiency**: hardware/software resources required

**Integrity**: deals with software system security

**Usability**: operation usability and training usability



# McCall model for software quality factors (2)



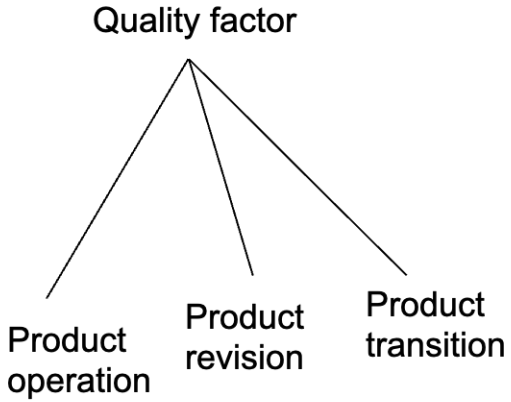

## Product revision factors

**Maintainability**: determines efforts needed by maintenance personnel to identify reasons of a software failure and completing correction.

**Flexibility**: capabilities and effort to support adaptive maintenance.

**Testability**: deals with testing process of a software system and its operation.

# McCall model for software quality factors (3)




## Product transition factors

**Portability**: relates to adaptation of system to other environments (hardware, operating system, etc.)

**Reusability**: use in future projects

**Interoperability**: requirement on creating interfaces with other software systems or equipment firmware

# Informal Quality Risk Analysis

## Quality Risk Analysis Table

| Risk category | Technical risk | Business risk | Priority | Testing tracking |
|---|---|---|---|---|
| Risk 1 | | | | |
| Risk 2 | | | | |
| Risk n | | | | |

**Technical level of risk**: likelihood that a fault might exist
assessment: system architect, designer, senior programmers

**Business level of risk**: impact of a given fault might have on the users, customers, and other stakeholders

Rating business and technical risks on some fixed scale:

1. very high, 2- high, 3. medium 4. low 5. very low

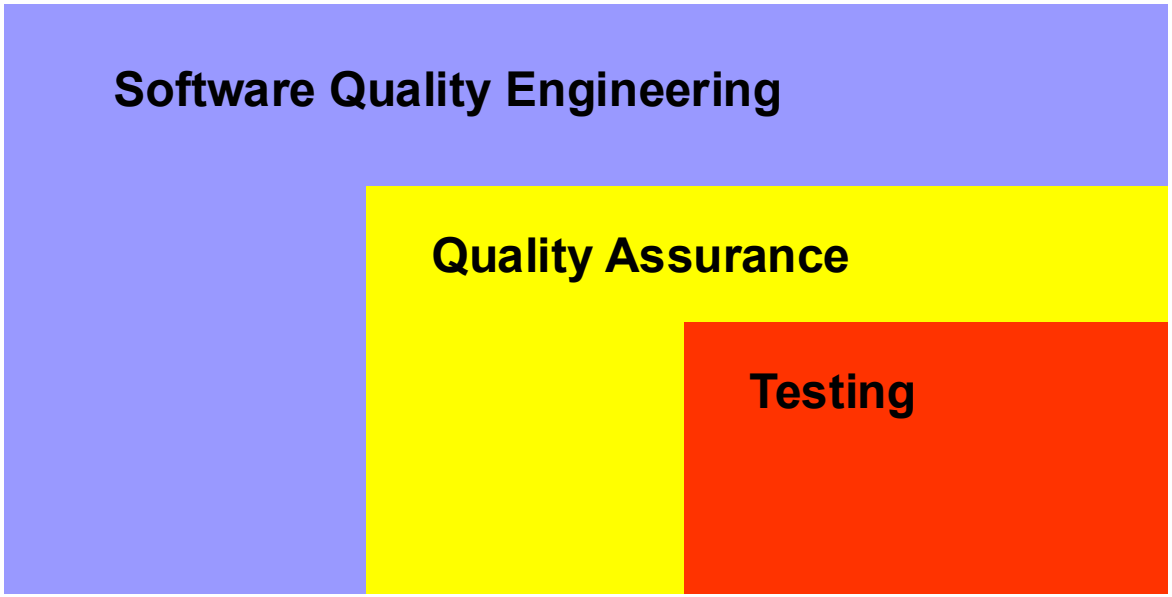
# Alternative Frameworks

**Different application domains – adoption of quality frameworks based upon specific business and market environments**

## CUPRIMDS @ IBM

**capability**
**usability**
**performance**
**reliability**
**installation**
**maintenance**
**documentation**
**service**

# Software Quality Engineering, Quality Assurance, and Testing: A hierarchy




**Quality assurance**: testing, inspection, formal verification, defect prevention, fault tolerance

# Errors, faults, failures

**IEEE Standard 610.12 (1990)**

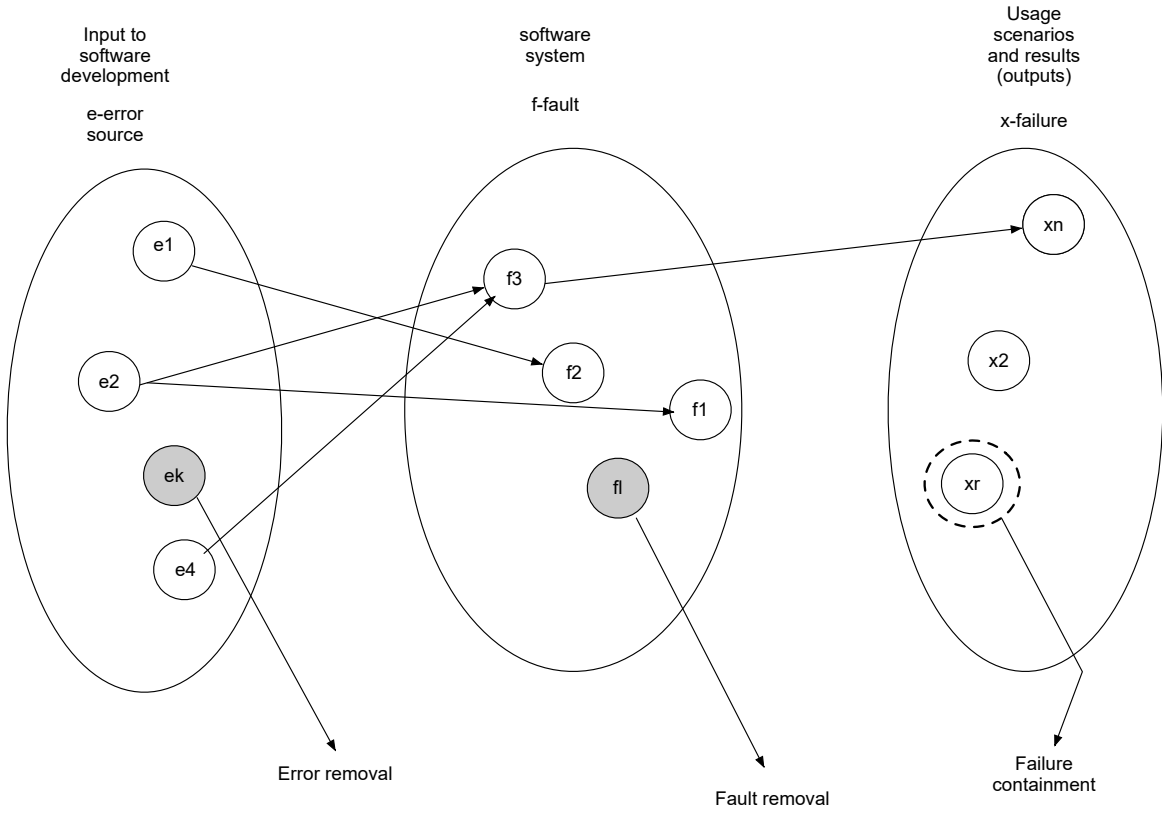
**Failure – the inability of system to perform its required functions within specified performance requirements**

**Fault – an incorrect step, process, or data definition in a computer program**

**Error – a human action that produces an incorrect result**

***Defects* = *{errors, faults, failures}***

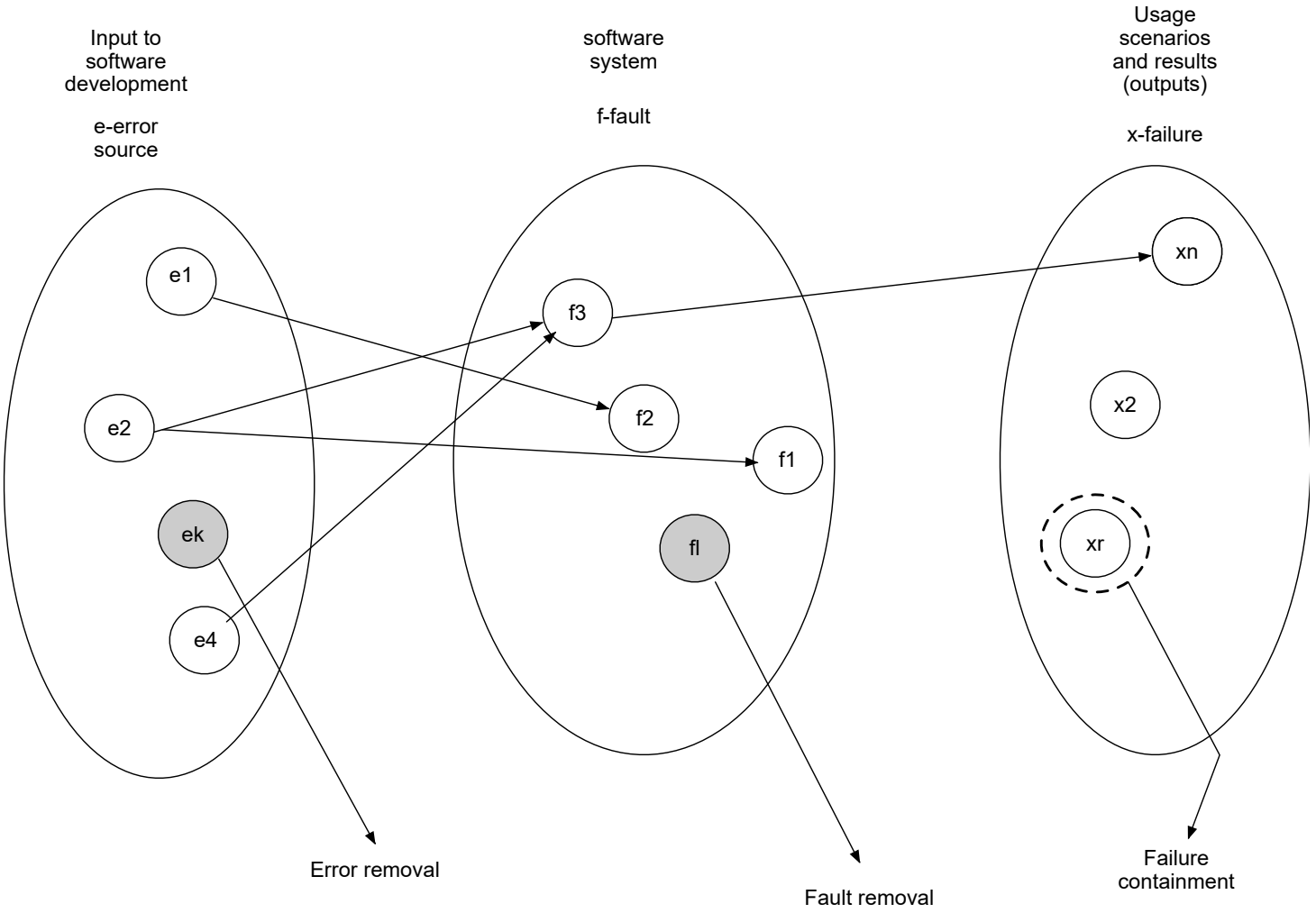
# Errors, faults, failures: relationships



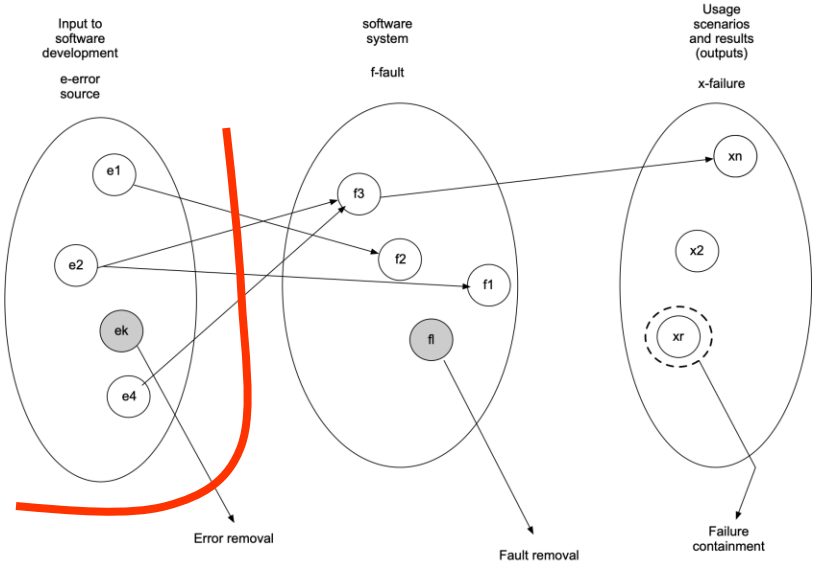

**Errors → faults → failures**

**Character of relationships: one-to-one, one-to-many…**

# Software quality assurance: Main ways of dealing with defects



# Software quality assurance: Main lines of defence




**Defect prevention**: error source elimination; education, training, formal specification and verification, selection of and application of appropriate technologies

# Software errors- causes (1)

**Definition of requirements**

**Client-developer communication failures**

**Deliberate deviations from software requirements**

**Logical design errors**

**Coding errors**

# Software errors- causes (2)

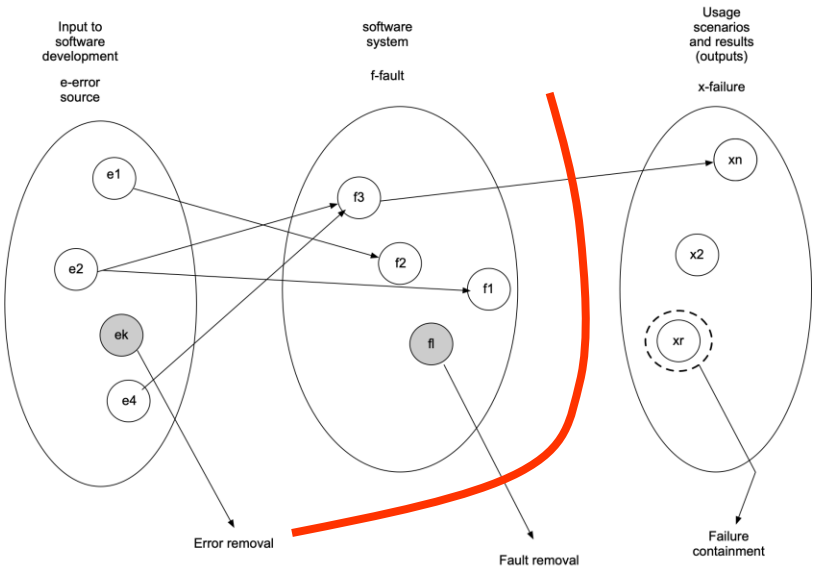
**Noncompliance with documentation and coding instructions**

**Shortcomings of the testing process**

**User interface and procedure errors**

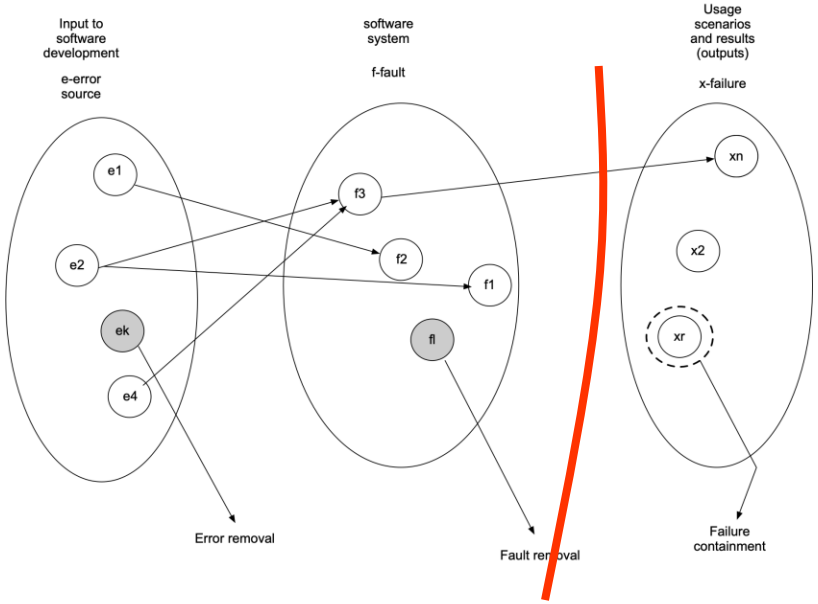
**Documentation errors**

# Software quality assurance: Main lines of defence



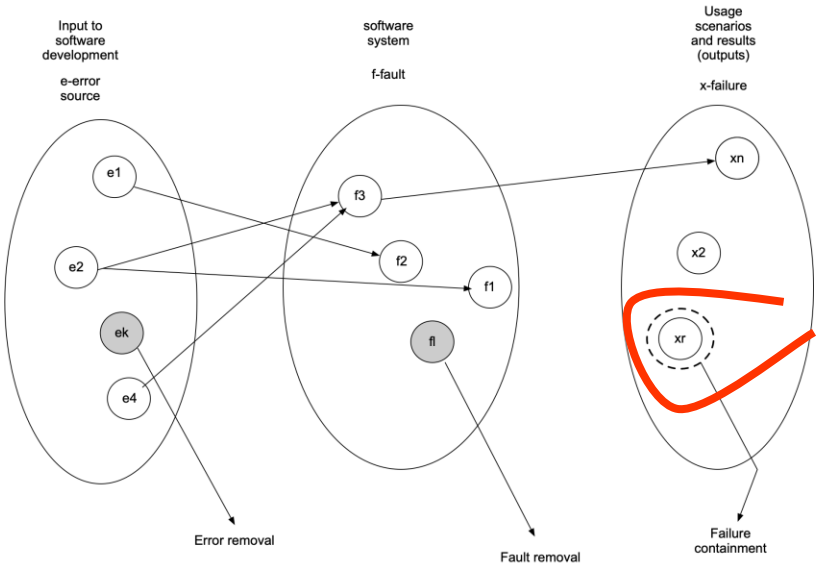

**Defect or fault removal : inspection and testing**

# Software quality assurance: Main lines of defence




**Failure prevention** : **fault tolerance, failure impact minimization [recovery blocks, N-version programming..]**

# Software quality assurance: Main lines of defence




**Defect containment :** failure impact minimization

# Verification and Validation (V & V) Perspective

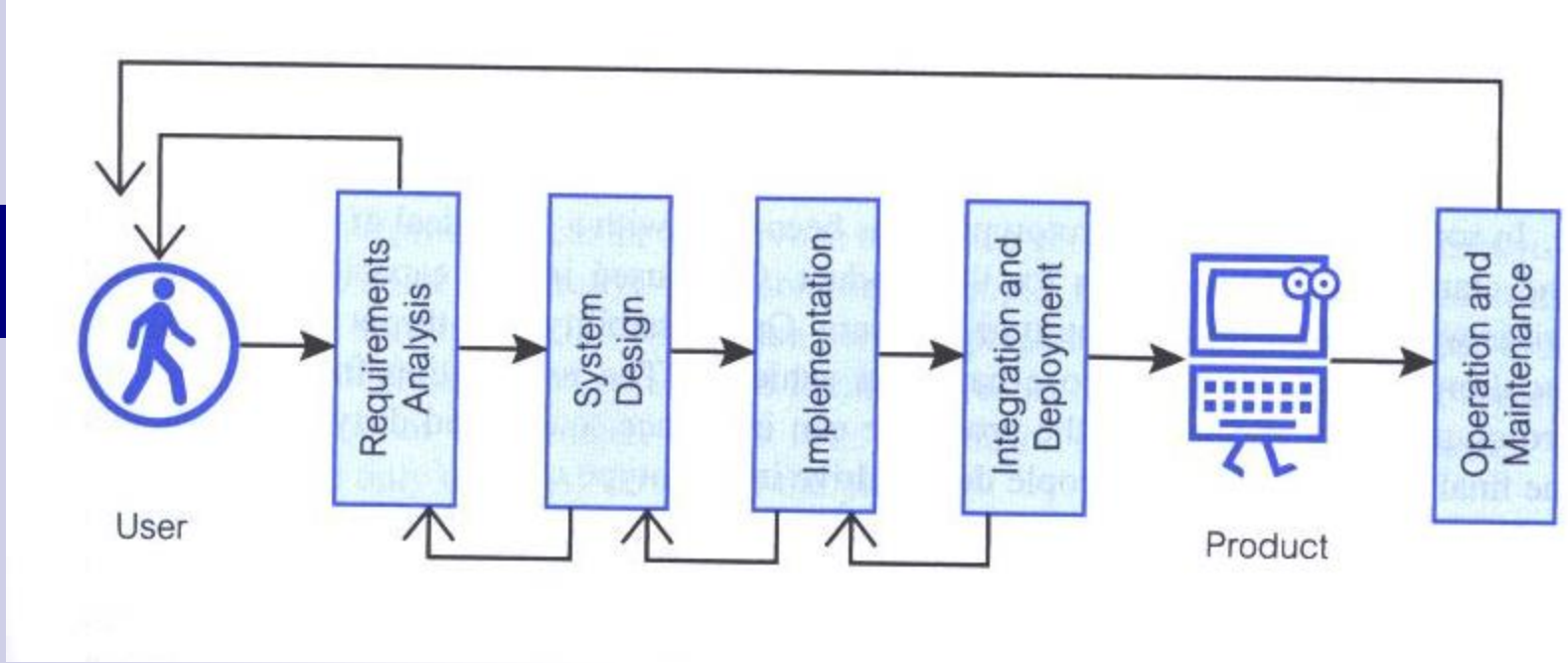
**Validation:** checks the conformance to the quality expectations of customers

deals directly with users and requirements

e.g., product specifications need to be validated through inspections and reviews

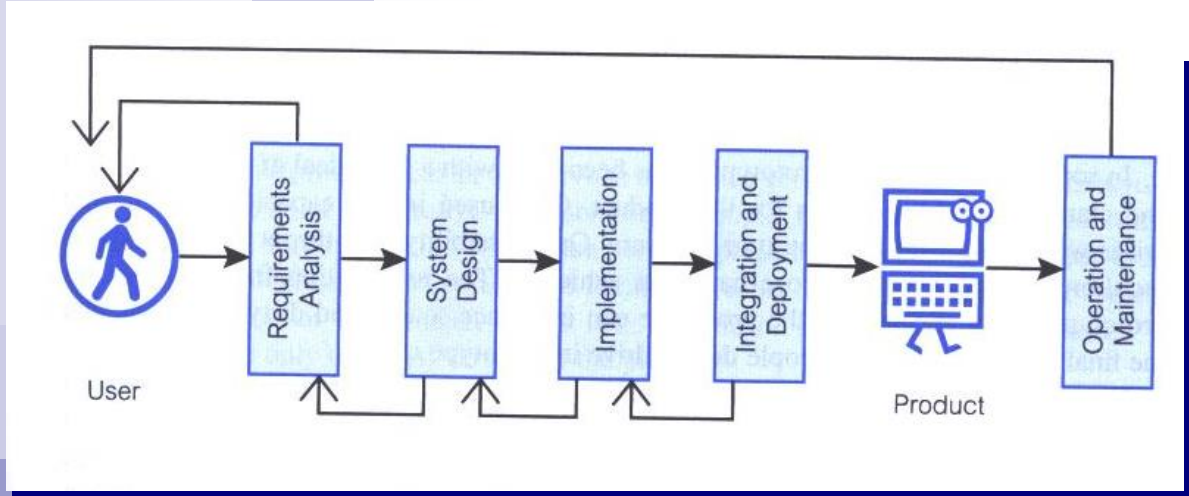
**Verification:** deals with internal product specifications

# Waterfall lifecycle




1970s
Large projects
Monolithic structure aimed at a single delivery
Linear sequence of phases
User involved only in early stages of requirements analysis

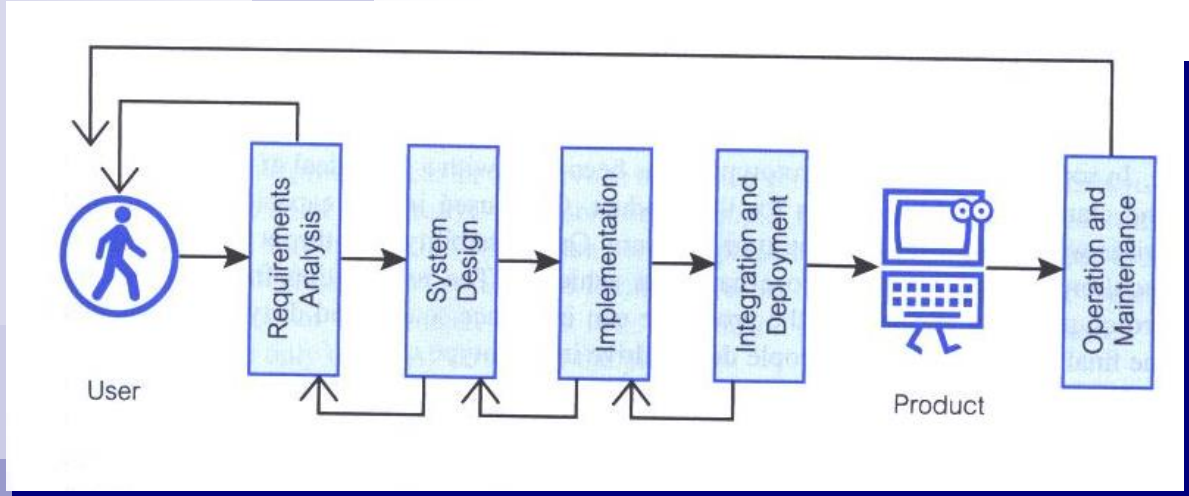
# Waterfall lifecycle




**Advantages**:

- enforces disciplined approach to software development.
- defines milestones in a clear manner
- facilitates project management
- produces complete documentation

# Waterfall lifecycle

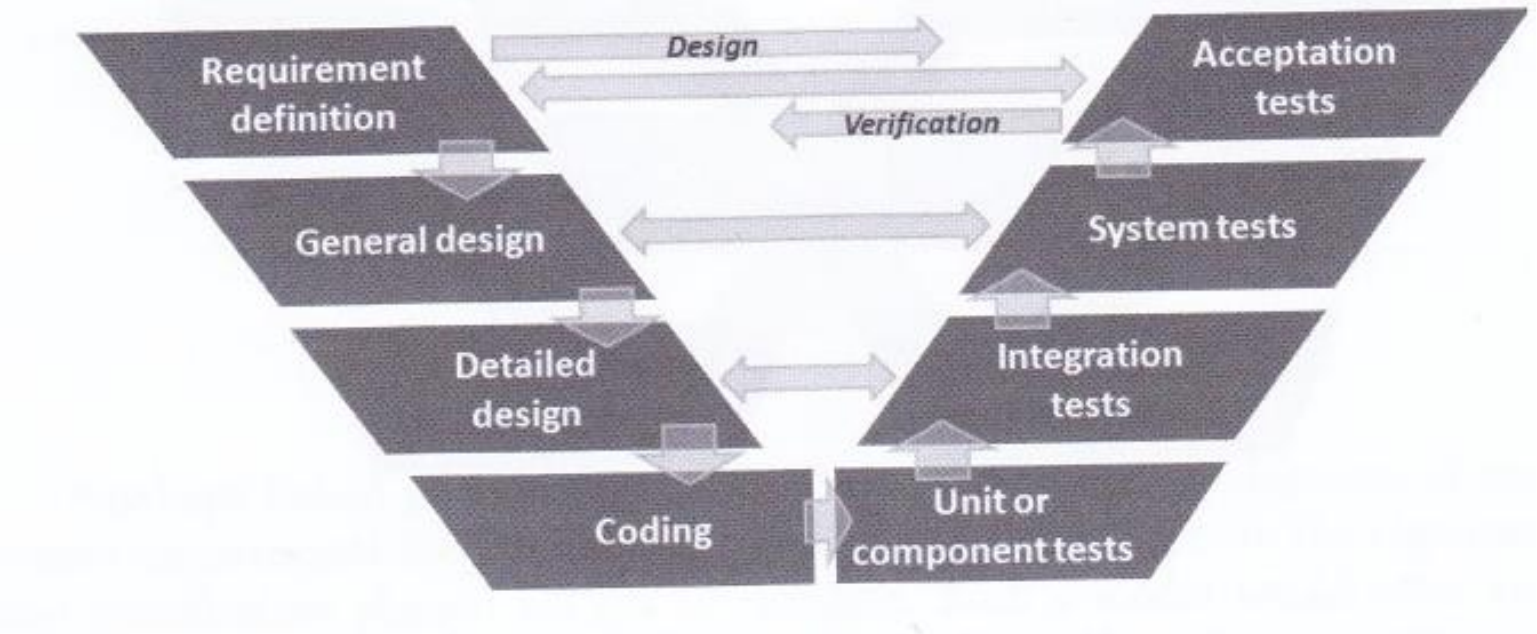



**Disadvantages**:

- monolithic approach – may take a lot of time to final product
- freezing the results of each phase (against principles of software engineering)
- project planning realized in early stages of the lifecycle

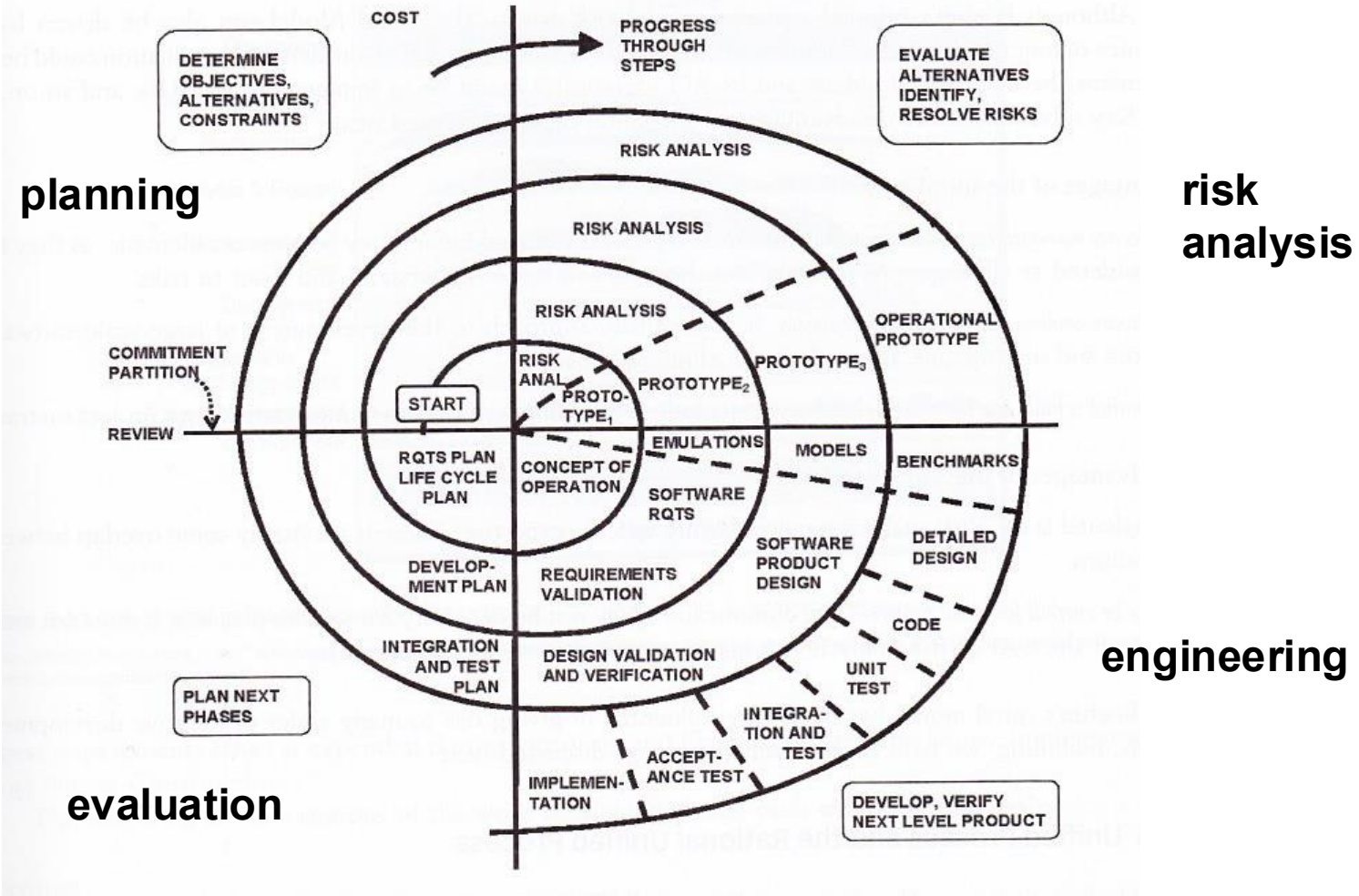
# V model

**V model links design activity to a test activity of the same level (main advantage of the model)**



# Spiral model (Boehm, 1988)

## Risk-driven process with incremental deliverables



# Spiral model (Boehm, 1988)

1. Identification of critical objectives and constraints of the product
2. Evaluation of project and process alternatives for achieving the objectives
3. Identification of risks
4. Cost effective resolution of a subset of risks
5. Development of project deliverables (requirements, design, implementation, testing)
6. Planning for next and future cycles
7. Stakeholders review of iteration deliverables

# Spiral model: advantages and disadvantages

**Advantages:**

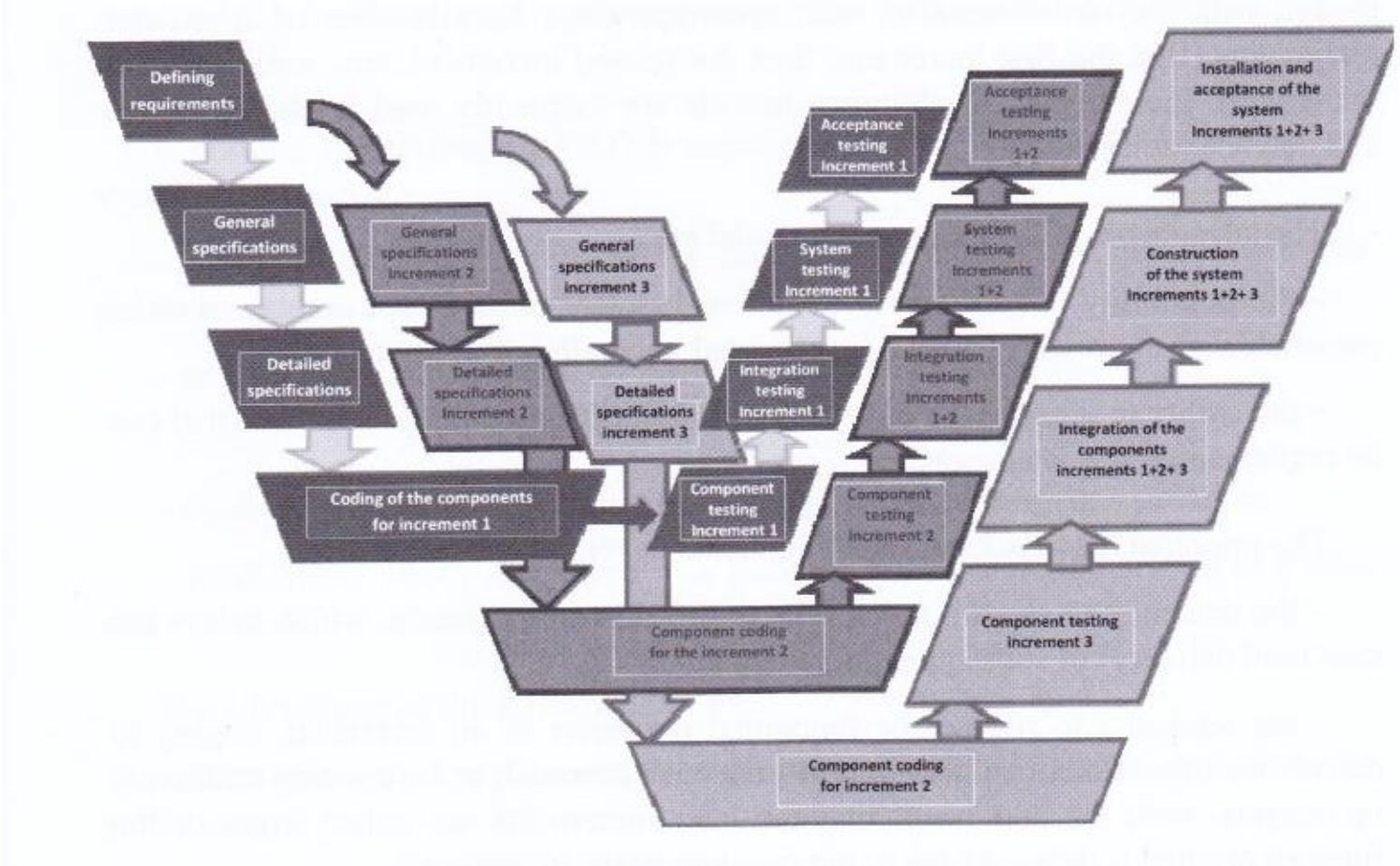
- Risk is managed early and throughout the process
- Software evolves as the project progresses; unattainable alternatives are eliminated early
- Planning is built into the process

**Disadvantages:**

- Complicated to use
- May be overkill for small projects

# Incremental model

- the design of an initial collection of components (functionalities) to which additional components are added until development is complete



# Incremental model: advantages and drawbacks

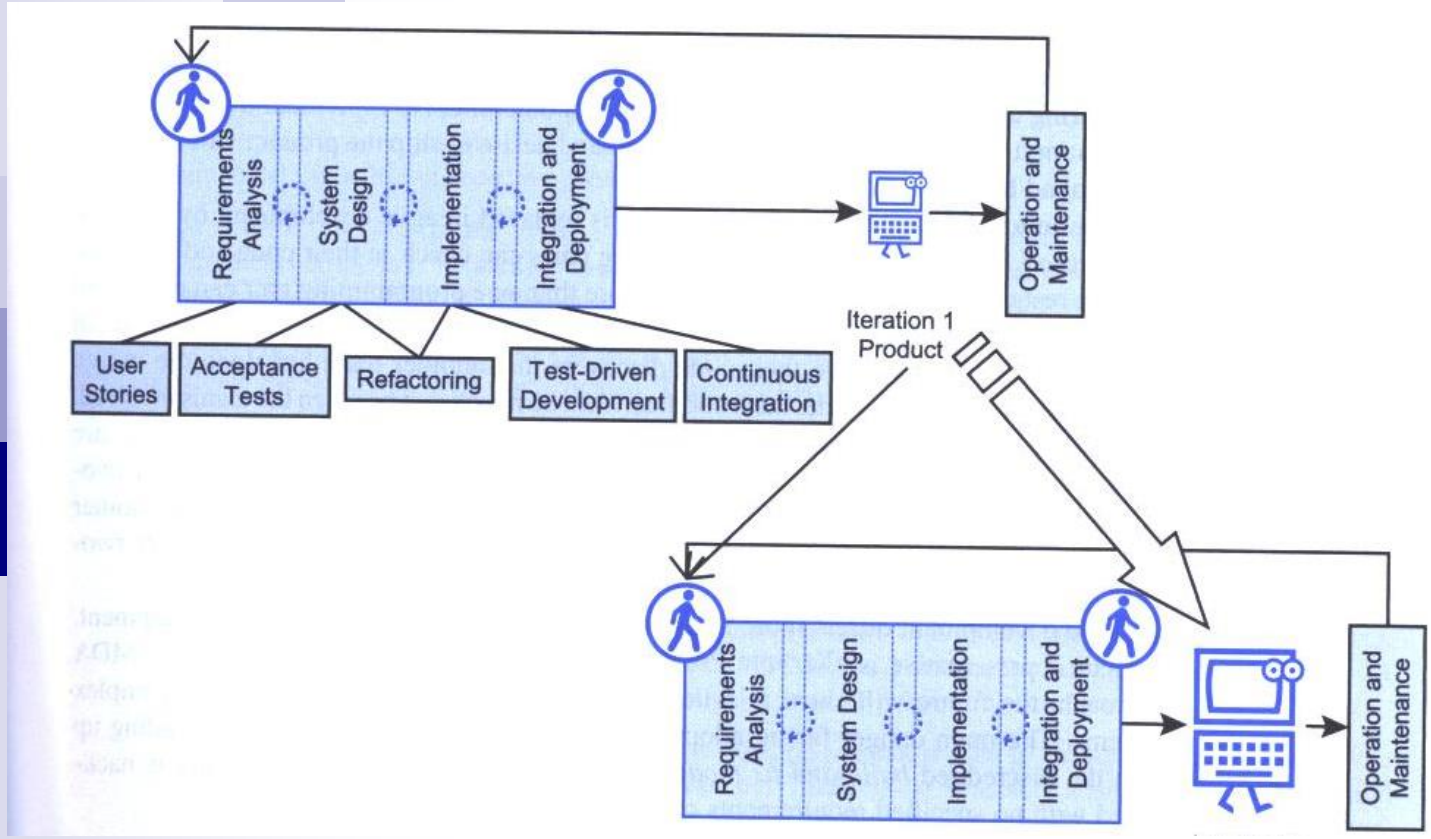
**Advantages:**

- Large development effort is split into a number of reasonably sized projects
- Identification of important critical components and those that can be implemented at later stage

**Disadvantages:**

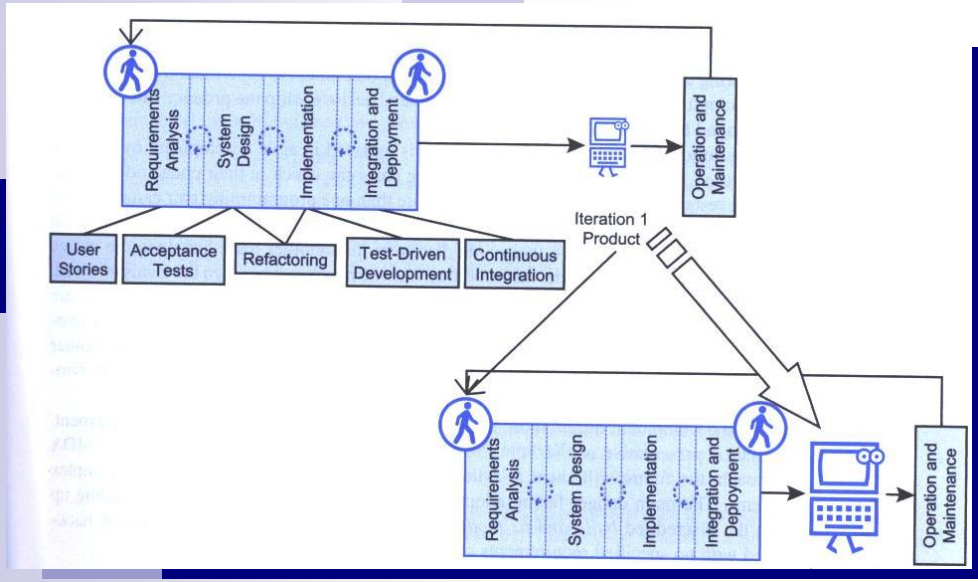
- The tendency to use too much time to develop an increment
- The tendency to reduce the functional perimeter of an increment, hoping to deliver the other functionalities in subsequent increments; later increments larger or the finished product delayed

# Agile lifecycle (2001)




**Agile Alliance (2001)- a new approach to software development**

# Agile lifecycle (2001)




**Manifesto of Agile Alliance (2001)**

1. **Individuals and interactions over processes and tools**
2. **Working software over comprehensive documentation**
3. **Customer collaboration over contract negotiations**
4. **Responding to change over following a plan**

# Agile lifecycle (2001)

**Software development is a creative activity that depends on people and team collaboration**

**In agile development, customers work closely with the development team**

**Pair programming, collective ownership, continuous integration and short cycles (2 week cycle-minor delivery, 6 week cycle- major delivery)**

# Agile lifecycle (2001)

**User stories**

Expressed by the customer to the developer

Natural language narrative:

- Behavior-driven development (BDD)
- Use cases (UML)

# Use cases

**user interaction with the system to realize a task**

Online Shopping
**Use Case:** "Purchase Items".
**Actor:** A customer.
**Goal:** To successfully buy products from an online store.
**Steps:** The customer selects a product, adds it to their cart, provides shipping and payment details, and receives a confirmation.

**Travel**
**Use Case:** "Book a Flight".
**Actor:** A traveler.
**Goal:** To reserve a seat on a flight.
**Steps:** The traveler searches for flights, selects their dates and destinations, provides passenger information, makes a payment, and receives a booking confirmation.

# Behavior-Driven Development (BDD)

**Scenario** 1: account is in credit

C1:given the account is in credit
C2:and the card is valid
C3:and the dispenser contains cash
C4:when the customer requests cash
A1:then ensure the account is debited
A2:and ensure cash is dispensed
A3:and ensure the card is returned

Short ID

IF (<pre-condition(s)>)
AND (data-condition(s)>)
AND (<input-event-sequence>)
THEN (<action sequence>)
AND (<output event sequence>)
AND(<post-condition(s)>)

Translation to decision tables

# Agile model: advantages and disadvantages

**Advantages:**

- The project always has demonstrable results
- Developers tend to be well motivated
- Customers are able to provide/refine requirements (based on the evolving product)

**Disadvantages:**

- Questionable in case of large applications
- Documentation problematic

# Development models: comparative analysis

| *Aspect* | **Waterfall** | **V-model** | **Incremental** | **Spiral** | **Agile** |
|---|---|---|---|---|---|
| Clear implementation for management | y | y | y | y | n |
| All requirements necessary before start | y | y | n/y | n | n |
| Ease of resources allocation | y | y | y | y | n |
| Quick availability of running software | n | n | y | y | y |
| Risk and cost control via prototyping | n | n | y | y | y |

# Testing: people and management

## Organizational activities

## People's skills

## Models

vertical model: organized around a product; one or more teams perform all different types of testing

horizontal model: mainly in large organizations; testing team performs one type of testing for many products