Introduction to Software Quality Assurance

Daniel Varró ♦ McGill University
daniel.varro@mcgill.ca

Based on: D. Galin ch.1-4.5 and R. Patton ch.1
Other sources: Stéphane S. Somé, Lionel Briand

Objectives

- What is **software quality**?
- Why is software quality important?
- What is **software quality assurance**?
- Software quality **factors**
- Elements of software quality assurance
- Development and quality **plans**
- Process **maturity** models

Table of Contents

What is Software and Software Engineering?

according to the IEEE:

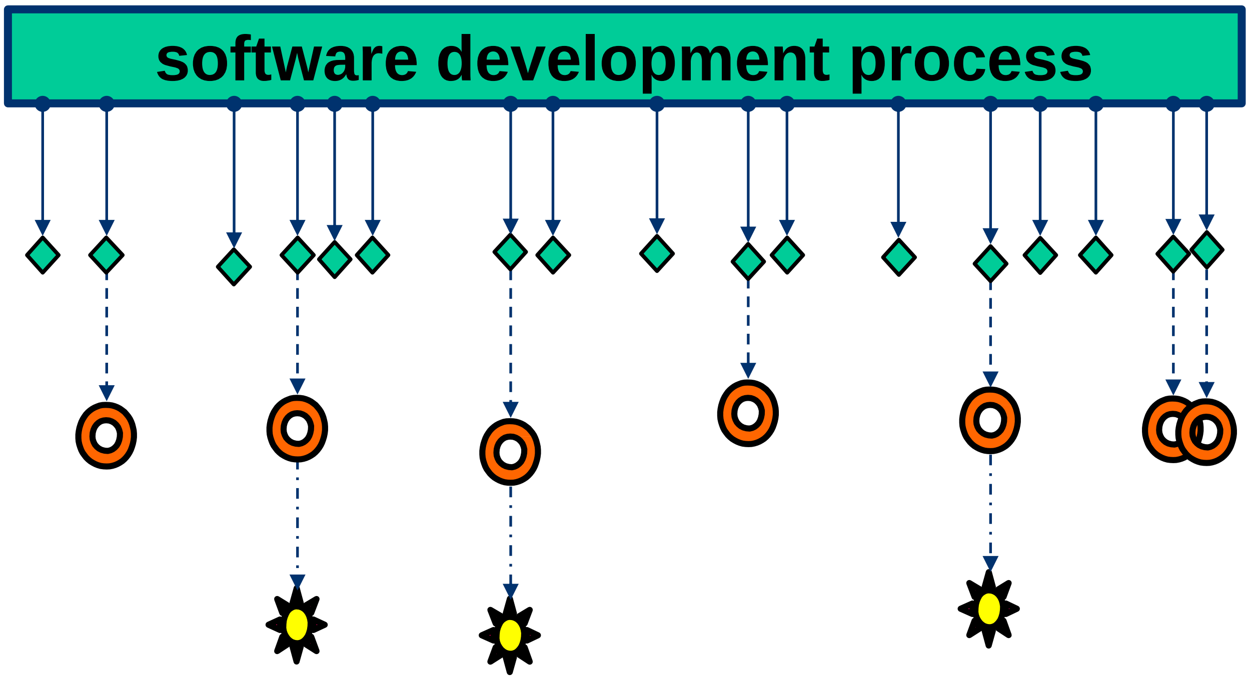
software = computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system

software engineering = (1) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software; (2) the study of approaches as in (1)

Software mistakes, Software defects/bugs, software failure

BASIC TERMINOLOGY

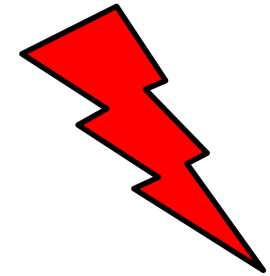
Software Error, Software Fault, and Software Failure



- **Bug/defect** consequence of a **human mistake**
- Results in non-conformance to requirements
- Manifests as **failure** in running software

Basic Testing Definitions

- **Mistake** ~~(error)~~: **people commit errors**
- **Defect (bug, ~~fault~~): a mistake (in the SW documentation, code, etc.) can lead to a defect**



Misleading terminology!

- **Failure: a failure occurs when a defect executes**
- **Incident: consequences of failures – failure occurrence may or may not be apparent to the user**
- **Software testing: exercise the software with test cases to gain (or reduce) confidence in the system (execution based on test cases)**
  - Expectation → reveal faults with failures incidences

Remember: Threats to Dependability

Source: I. Majzik (BME, Hungary)

Fault → Error → Failure

- Fault: Adjudged or hypothesized cause of an error
- Error: Erroneous state of a component leading to a failure
- Failure: Delivered service deviates from correct service

	Fault	Error	Failure
HW	Bit flip in memory due to a cosmic particle	Reading the faulty memory cell results in incorrect value	The robot arm collides with the wall
SW	Programmer increases a variable instead of decreasing it	The value of the variable will be incorrect when the faulty statement executes	The final result of computation will be incorrect

Defects: Root Cause and Effect

- **Root cause of a defect:**
  - Earliest action or condition that contributed to creating the defect
- **Root cause analysis:**
  - Identify root cause to reduce occurrence of similar defects in the future
- **Effect of a defect:**
  - Observed by user / customer, product owner (PO)

- **Example:**
  - Root cause: PO misunderstands how to calcute interest rates ➔ Ambiguous user story ➔ Wrong calculation in code ➔ Incorrect interest payment ➔ Customer complaint :Effect

Nine Causes of Software Defects

1) Faulty requirements definition (incorrect, missing, incomplete, unnecessary requirements)

2) Client–developer communication failures

3) Deliberate deviations from software requirements (improper reuse, omission due to time pressure, gold-plating)

4) Logical design errors (problems with algorithms, sequencing of actions, boundary conditions, missing states, how to handle “illegal” input)

Nine Causes of Software Defects

5) Coding errors

6) Non-compliance with documentation and coding instructions (more difficult to deal with team attrition and inspections)

7) Shortcomings of the testing process

8) Procedure errors (workflow and user interface errors)

9) Documentation errors

Exercise: Procedure Error correct?

Eiffel decides to grant a 5% discount at the beginning of the month to those customers with total purchases in excess of $1 million in the last 12 months except for those customers that returned in excess of 10% of their purchases during the last three months.

At the end of each year, Eiffel's central information processing department:
(1) Collects the previous year's sales data for each of the customers from all the chain's stores.
(2) Calculates the cumulative purchases of each customer for the previous year in all the chain's stores.
(3) Prepares a list of all customers whose purchases exceed $1 million and distribute it to all stores.

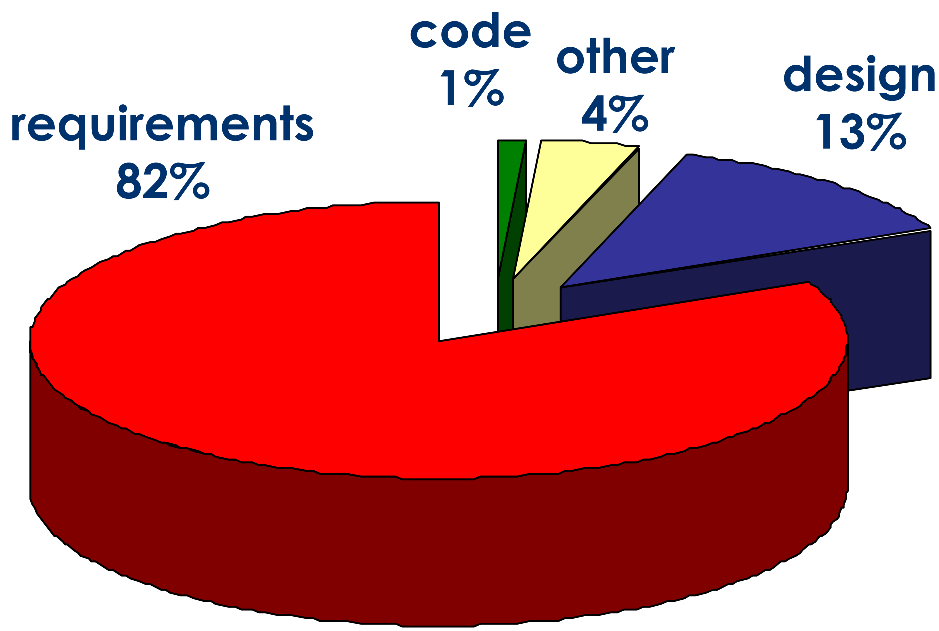
At the end of each quarter, the individual store's information processing unit:
(1) Calculates the percentage of goods returned during the last quarter for each customer.
(2) Prepares a list of all customers whose returned goods for the last quarter exceed 10% of that quarter's purchase.

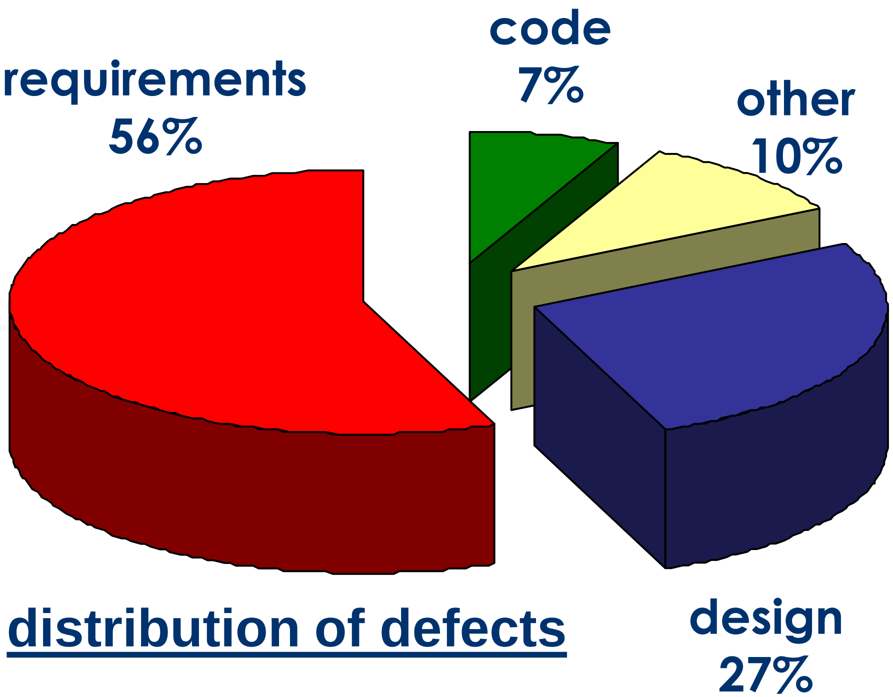
At the beginning of the month, the store's information processing unit:
(1) Processes all monthly purchases for each of the customers.
(2) Calculates the discount according to the last year's purchase data in all stores, and according to the store's records of returns in the last quarter.

Development Phases vs. Defects

- majority of defects are introduced **in earlier phases**

distribution of effort to fix defects





distribution of defects

- **requirements** are the top reason for project success or failure

Source: Martin & Leffinwell

Development Phases vs. Cost

relative cost of fixing defects

phase in which found	**cost ratio**
requirements	1
design	3 – 6
coding	10
unit/integration testing	15 – 40
system/acceptance testing	30 – 70
production	40 – 1000

IMPORTANCE OF SOFTWARE QUALITY

What is Software Quality?

- **Conformance to requirements:**
  - Lack of bugs
    - Low defect rate (# of defects/size unit, e.g. #bugs/LOC)
    - Well-documented defects (known issues)
  - High reliability / availability
    (number of failures per *N* hours of operation)
    - Mean time to failure (MTTF), i.e., the probability of failure-free operation until a specified time
    - Mean time between failures (MTBF), i.e. the probability that the system is up and running at any given point in time

What is Software Quality?

according to the IEEE:

software quality = (1) the degree to which a system, component, or process meets specified requirements; (2) the degree to which a system, component, or process meets customer or user needs or expectations

according to Pressman:

software quality = conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software

Importance of Software Quality

- **Software is a major component of computer systems (about 80% of the cost) used for**
  - Communication (e.g., phone system, email system)
  - Health monitoring
  - Transportation (e.g., automobile, aeronautics),
  - Economic exchanges (e.g., e-commerce),
  - Entertainment,
  - etc.

- **Software defects may be extremely costly in terms of**
  - Money
  - Reputation
  - Loss of life

Importance of Software Quality

- **Zune 30 leap year freeze:**

- **On December 31st 2008, players began freezing at about midnight becoming totally unresponsive and practically useless**

- **Official fix:**
  - Wait until January 1st 2009

Importance of Software Quality

- **Zune 30 leap year freeze:**

```
/* days: the number of days since January 1, 1980. */
year = ORIGINYEAR; /* = 1980 */
while (days > 365)
{
        if (IsLeapYear(year))
        {
                if (days > 366)
                {
                        days -= 366;
                        year += 1;
                }
        }
        else
        {
                days -= 365;
                year += 1;
        }
}
```

source of the problem?

Importance of Software Quality

- **iPhone alarm glitch:**
- **Reoccurring alarm on some devices using the mobile operating system iOS 4.1 failed to properly work after daylight saving time switch**
- **Problem repeated on January 1st 2011 – this time for non-reoccurring alarms**

Importance of Software Quality

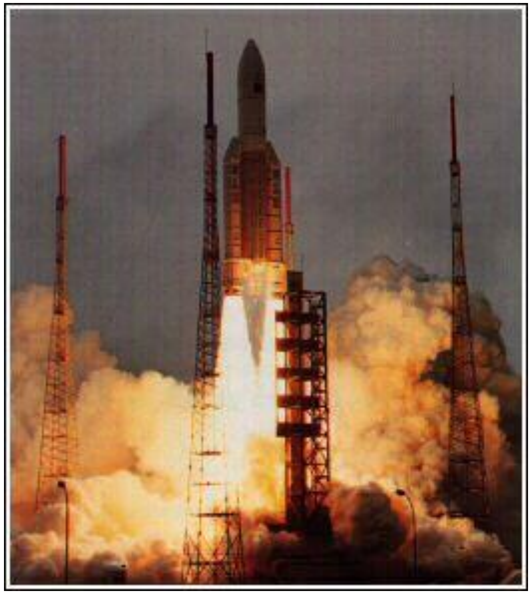
- **Several historic disasters attributed to software:**
  - 1988 shooting down of Airbus 320 by the USS Vincennes – cryptic and misleading output displayed by tracking software
  - 1991 patriot missile failure – inaccurate calculation of time due to computer arithmetic errors
  - London Ambulance Service Computer Aided Dispatch System – several deaths
  - Therac-25: radiation therapy and X-ray machine killed several patients; cause: unanticipated, non-standard user inputs

Importance of Software Quality

- **Several historic** **disasters** **attributed to software: (cont'd)**
  - On June 3, 1980, the North American Aerospace Defense Command (NORAD) reported that the U.S. was under missile attack
  - First operational launch attempt of the space shuttle, whose real-time operating software consists of about 500,000 lines of code, failed – synchronization problem among its flight-control computers
  - 9 hour breakdown of AT&T's long-distance telephone network – caused by an untested code patch

Importance of Software Quality

- **Ariane 5 crash – June 4, 1996:**
  - Maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff
  - Loss was about half a billion dollars
  - Explosion was the result of a software error
  - Uncaught exception due to floating-point error: conversion from a 64-bit integer to a 16-bit signed integer applied to a larger than expected number



Importance of Software Quality

- **Ariane 5 crash – June 4, 1996: (cont'd)**
  - Runtime error (out of range, overflow) was detected and computer shut itself down
  - Same for the backup computers
  - This resulted in the total loss of attitude control
  - Ariane 5 turned uncontrollably and aerodynamic forces broke the vehicle apart
  - Breakup was detected by an on-board monitor which ignited the explosive charges to destroy the vehicle in the air

  - Ironically, the result of the format conversion was no longer needed after lift off

Importance of Software Quality

- **Ariane 5 crash – June 4, 1996: (cont’d)**
  - Module was reused without proper testing from Ariane 4
  - Error was not supposed to happen with Ariane 4 (it was shown that such a large input could not occur in the context of Ariane 4, no exception handler)
  - Note this was not a complex, computing problem, but a deficiency of the software engineering practices in place …

Importance of Software Quality

- **Mars Climate Orbiter – September 23, 1999:**
  - Disappeared as it began to orbit Mars
  - Cost about $US 125 million
  - Failure due to error in a transfer of information between a team in Colorado and a team in California
  - One team used English units (e.g., inches, feet, and pounds) while the other used metric units for a key spacecraft operation

Importance of Software Quality

- **Mars Polar Lander – December, 1999:**
  - Disappeared during landing on Mars
  - Failure most likely due to unexpected setting of a single data bit
  - Defect not caught by testing
  - Independent teams tested separate aspects

- **More software failure examples: http://www.cse.lehigh.edu/~gtan/bug/softwarebug.html**

Importance of Software Quality

- **Internet viruses and worms:**
  - Blaster worm ($US 525 millions)
  - Sobig.F ($US 500 millions – 1 billion)

  - Exploit well known software vulnerabilities

  - Software developers do not devote enough effort to applying lessons learned about the causes of vulnerabilities

  - Same types of vulnerabilities continue to be seen in newer versions of products that were in earlier versions

Importance of Software Quality

- **The Heartbleed Bug:**
  - Serious vulnerability in the popular OpenSSL cryptographic software library
  - Potentially affected open source web servers like Apache and nginx with a combined market share of over 66%; plus email servers, chat servers, and VPNs
  - Out in the wild since March 14, 2012 – fixed April 7, 2014
  - August 2014: personal data of 4.5 million patients of U.S. hospital group Community Health Systems Inc. stolen by exploiting the Heartbleed bug

http://heartbleed.com/

Importance of Software Quality

- **Pervasive problems:**
  - Software is commonly delivered late, way over budget, and of unsatisfactory quality
  - Software validation and verification are rarely systematic and are usually not based on sound, well-defined techniques
  - Software development processes are commonly unstable and uncontrolled
  - Software quality is poorly measured, monitored, and controlled

Importance of Software Quality

- **Monetary impact of poor software quality (Standish group – 1995):**
  - 175,000 software projects/year – average cost / project
    - Large companies: $US 2,322,000
    - Medium companies: $US 1,331,000
    - Small companies: $US 434,000
  - 31% of projects canceled before completed (cost $81 billion)
  - 53% of projects exceed their budget, costing 189% of original estimates (cost $59 billion)

Importance of Software Quality

- **Monetary impact …: (cont'd)**
  - 16% of software projects completed on-time and on-budget (9% for larger companies)
  - Large companies: delivered systems have approximately only 42% of originally-proposed features and functions
  - Smaller companies: 78% of projects get deployed with at least 74% of their original features and functions
- **~~NIST study (2002): bugs cost US economy $ 59.5 billion~~ a year—earlier detection could save $22 billion**

The Software Quality Challenge

- **The uniqueness of the software product:**
  - High complexity (and increasingly so) – pervasive in an increasing number of industries
  - Invisibility of the product
  - Limited opportunities to detect defects compared to other industries
    - Development, not production (only opportunity to detect defects is product development; product production planning not required; simple manufacturing)

The Software Quality Challenge

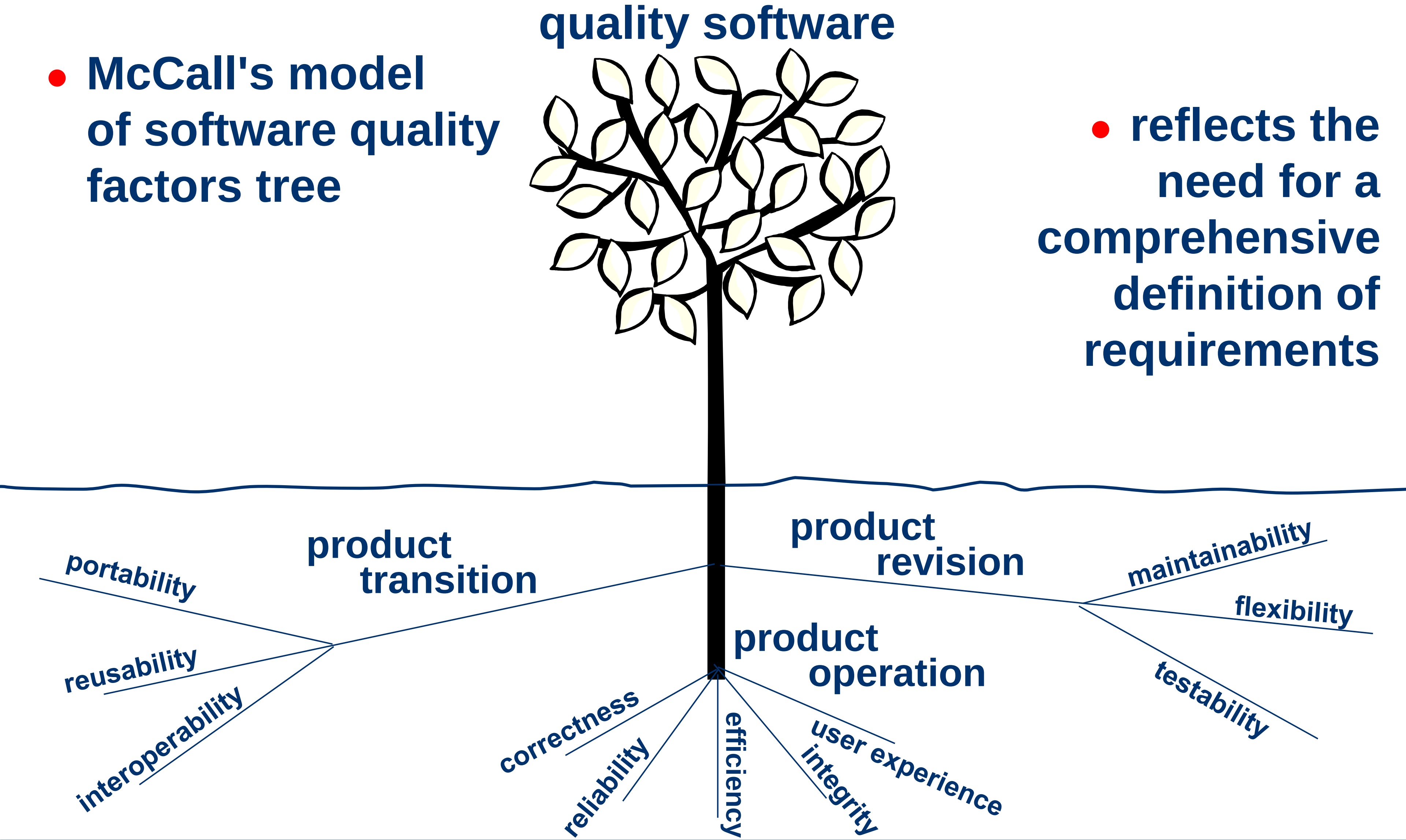
- **The environments in which software is developed:**
  - Contracted (features / budget / timetable)
  - Subjection to customer-supplier relationship (potential miscommunications, change request management)
  - Requirement for teamwork (human-intensive; engineering but also a social process)
  - Need for cooperation and coordination with other development teams
  - Need for interfaces with other software systems
  - Need to continue carrying out a project while the team changes
  - Need to continue maintaining the system for years

SOFTWARE QUALITY FACTORS

Software Quality Factors

- McCall's model of software quality factors tree

- reflects the need for a comprehensive definition of requirements



Software Quality Factors

- **Correctness**
  - Accuracy and completeness of required output
  - Up-to-dateness and availability of the information
- **Reliability / Availability**
  - First failure / Maximum failure rate
- **Efficiency**
  - Hardware resources needed to perform software function (processing capabilities, data storage, bandwidth, power usage)
- **Integrity**
  - Software system security, access rights

Software Quality Factors

- **Usability / User Experience**
  - How intuitive is it to use the software?
  - How much training required (to learn and perform required task)?
- **Maintainability**
  - Effort to identify and fix software failures (modularity, documentation, etc)
- **Flexibility**
  - Degree of adaptability (to new customers, tasks, etc)
- **Testability**
  - Support for testing (e.g., log files, automatic diagnostics, etc), traceability

Software Quality Factors

- **Portability**
  - Adaptation to other environments (hardware, software)
- **Reusability**
  - Use of software components for other projects
- **Interoperability**
  - Ability to interface with other components/systems

- **Other factors: robustness, performance, user friendliness, verifiability, repairability, evolvability, understandability, safety, manageability**

SOFTWARE QUALITY ASSURANCE

What is Software Quality Assurance?

according to the IEEE:

software quality assurance = (1) a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements; (2) a set of activities designed to evaluate the process by which the products are developed or manufactured – contrast with: quality control

quality control: set of activities designed to evaluate the quality of a developed or manufactured product – after development before shipment

quality assurance aims to minimize the cost of guaranteeing quality

What is Software Quality Assurance?

according to D. Galin:

software quality assurance = a systematic, planned set of actions necessary to provide adequate confidence that the software development process or the maintenance process of a software system product conforms to established functional technical requirements as well as with the managerial requirements of keeping the schedule and operating within the budgetary confines

Objectives of Software Quality Assurance (SQA)

1) Assuring an acceptable level of confidence that the software will conform to **functional** technical requirements

2) Assuring an acceptable level of confidence that the software will conform to managerial **scheduling** and **budgetary** requirements

3) Initiation and management of **activities** for the improvement and greater efficiency of software development, software maintenance, and software quality assurance activities

Three General Principles of Software Quality Assurance

- Know what you **are** doing
- Know what you **should be** doing
- Know **how to measure** the difference

3 General Principles of SQA

- **Know what you are doing:**
  - Understand what is being built, how it is being built and what it currently does
  - Implies a software development process with
    - Management structure (milestones, scheduling)
    - Reporting policies
    - Tracking

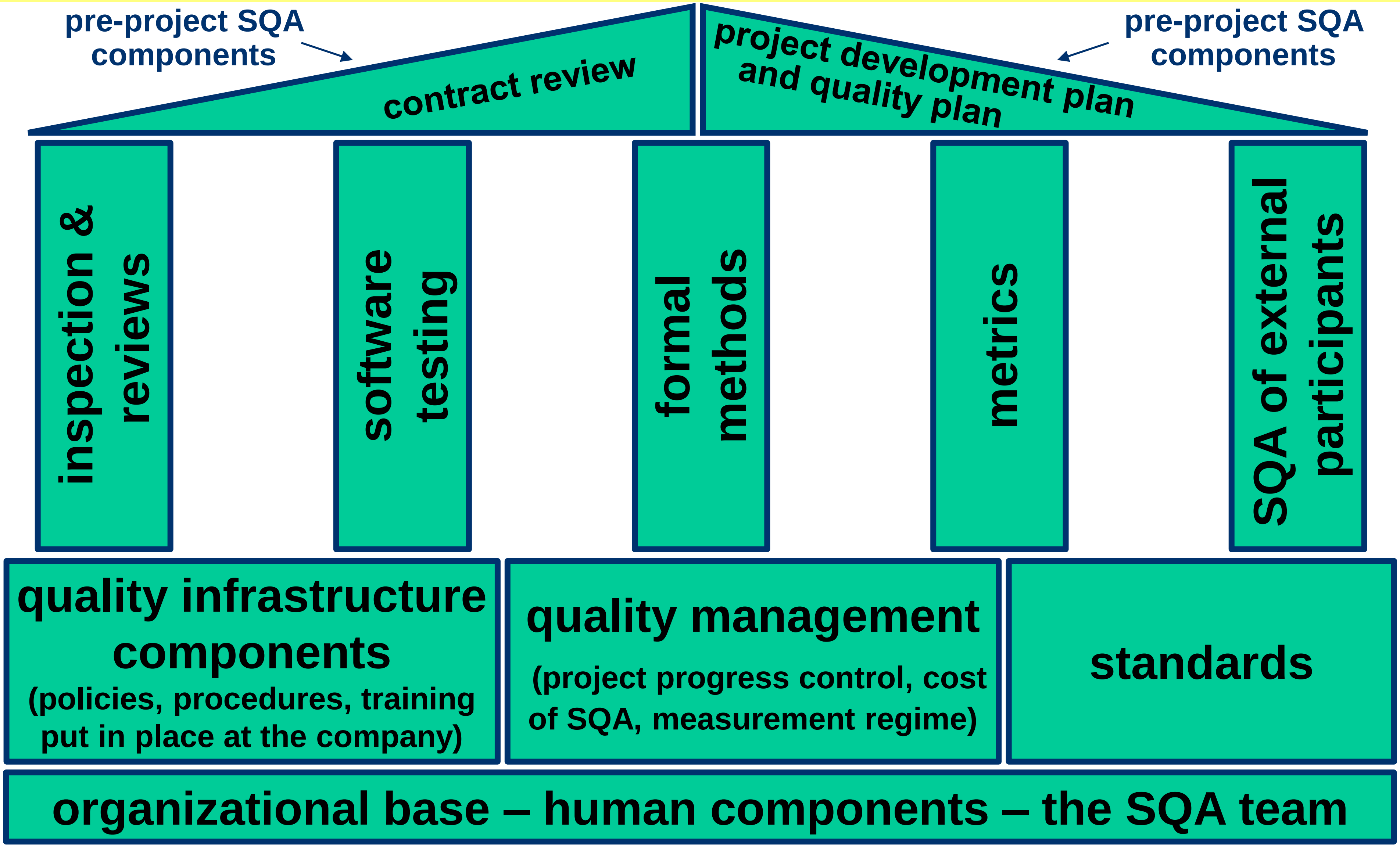
3 General Principles of SQA

- **Know what you should be doing:**
  - Having explicit requirements and specifications
  - Implies a software development process with
    - Requirements analysis
    - Acceptance tests
    - Frequent user feedback

3 General Principles of SQA

- **Know how to measure the difference:**
  - Having explicit measures comparing what is being done with what should be done
  - Four complementary methods:
    1) **Formal methods – verify mathematically specified properties**
    2) **Testing – explicit input to exercise software and check for expected output**
    3) **Inspections – human examination of requirements, design, code, … based on checklists**
    4) **Metrics – measure a known set of properties related to quality**

“The Software Quality Shrine”



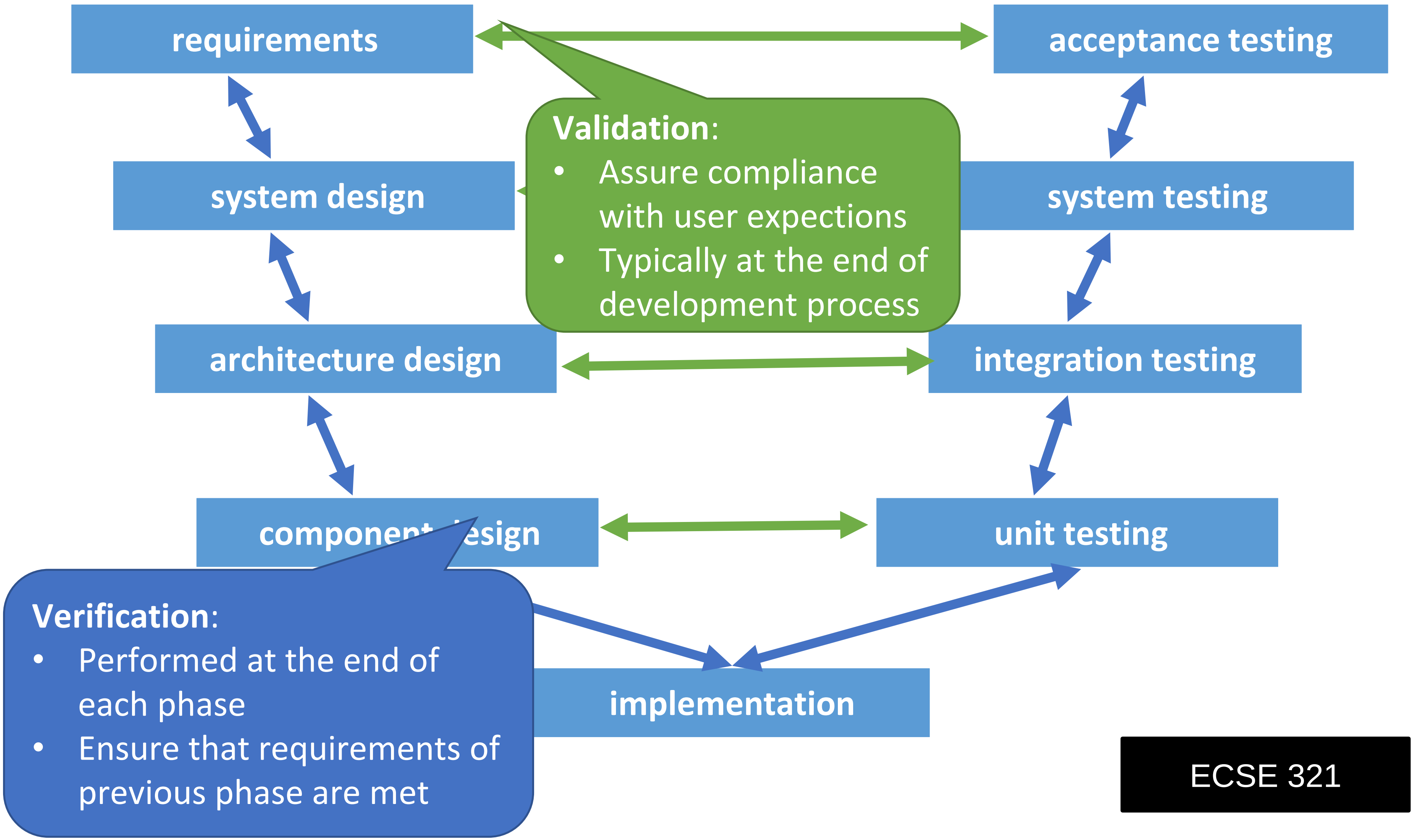
Software Quality Assurance

- **SQA is a comprehensive lifecycle approach concerned with every aspect of the software product development process**
- **Includes:**
  - Comprehensive set of quality objectives
  - Measurable quality attributes (quality metrics) to assess progress toward the objectives
  - Quantitative certification targets for all component of the software development processes
- **Takes into account:**
  - Customer product requirements
  - Customer quality requirements
  - Corporate quality requirements

Software Quality Assurance

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

Verification & Validation in SQA



Software Quality Assurance

- **SQA includes:**

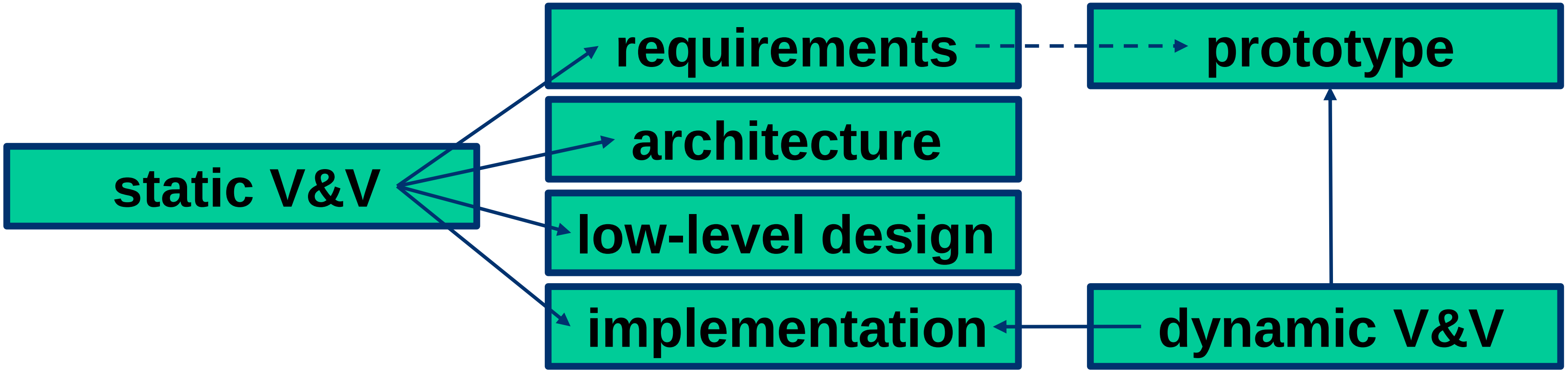
- **Defect prevention**
  - Prevents defects from occurring in the first place
  - Activities: training, planning, and simulation

- **Defects detection**
  - Finds defects in a software artifact
  - Activities: inspections, testing, or measuring

- **Defects removal**
  - Isolation, correction, verification of fixes
  - Activities: fault isolation, fault analysis, regression testing

Software Quality Assurance

- **Typical activities of an SQA process:**
  - Requirements validation
  - Design verification
  - Static code checking (inspection/reviews)
  - Dynamic testing
  - Process engineering and standards
  - Metrics and continuous improvement



Traits of Software Testers

explorers

troubleshooters

relentless

creative

(mellowed) perfectionists

exercise good judgement

tactful and diplomatic

persuasive

Key SQA Capabilities

- Uncover faults in the documents **where they are introduced**, in a systematic way, in order to avoid ripple effects – **systematic, structured reviews** of software documents are referred to as **inspections**
- Monitor and control **quality**, e.g., reliability, maintainability, safety, across **all** project phases and activities
- Derive, in a **systematic** way, effective test cases to uncover faults
- **Automate testing** and **inspection** activities, to the maximum extent possible
- All this implies the **measurement** of software products and processes and the **empirical evaluation** of testing and inspection technologies

Sec. 2.1: Foundation Level Syllabus of ISTQB

SOFTWARE DEVELOPMENT LIFECYCLE MODELS

Sequential vs. Iterative processes

- **Sequential**
  - e.g. Waterfall, V-model
  - deliver software that contains the complete set of features, but typically require months or years for delivery
- **Iterative**
  - E.g. Rational Unified Process, Kanban, Scrum, Spiral
  - Establish requirements, designing, building, and testing a system in pieces
  - Software features grow incrementally
- **Selection of process is context dependent**
  - Safety critical systems vs. Mobile apps
  - IoT: different SW development process for each device
- **Recap from ECSE 321**

Continuous … and Testing

- **Continuous Integration (CI)**
  - A software development process where a continuous integration server rebuilds a branch of source code every time code is committed to the source control system
  - The process is often extended to include deployment, installation, and testing of applications in production environments
- **Continuous Deployment**
  - A software production process where changes are automatically deployed to production without any manual intervention
- **Continuous Delivery**
  - A software production process where the software can be released to production at any time with as much automation as possible for each step

(see addendums to lecture notes from DZone)

Closing Thought & Discussion

"Testing by itself does not improve software quality. Test results are an indicator of quality, but in and of themselves, they don't improve it. Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often. What you eat before you step onto the scale determines how much you will weigh, and the software development techniques you use determine how many errors testing will find. If you want to lose weight, don't buy a new scale; change your diet. If you want to improve your software, don't test more; develop better."

Steve McConnell, Code Complete