POLITECNICO
MILANO 1863

Software Engineering

Definitions
The process and the product
Phases of the development process

M Camilli, E Di Nitto, M Rossi

Why software engineering is important

- Software is everywhere and our society is now totally dependent on software-intensive systems
- Society could not function without software
  - Healthcare Systems
  - Transportation Systems
  - Energy Systems
  - Manufacturing Systems
  - …
- Software failures cannot be tolerated… but they still happen!

Example: worldwide IT outage on July 19, 2024

- CrowdStrike crashed enterprise Windows-based systems worldwide (approximately 8.5 million devices)
  - Airlines including American Airlines, Delta, KLM, Lufthansa, Ryanair, SAS, and United
  - Airports (e.g., Gatwick, Luton), healthcare, and media organizations (e.g., MTV, Sky, BBC)
  - Financial organizations including the London Stock Exchange, Lloyds Bank, and Visa
  - Sporting bodies including F1 teams (e.g., Aston Martin, Mercedes, and Williams) and the Paris 2024 Olympic Organizing Committee
- Array out of bounds read issue!
  - https://www.computerweekly.com/feature/CrowdStrike-update-chaos-explained-What-you-need-to-know
  - https://www.crowdstrike.com/wp-content/uploads/2024/08/Channel-File-291-Incident-Root-Cause-Analysis-08.06.2024.pdf

:(

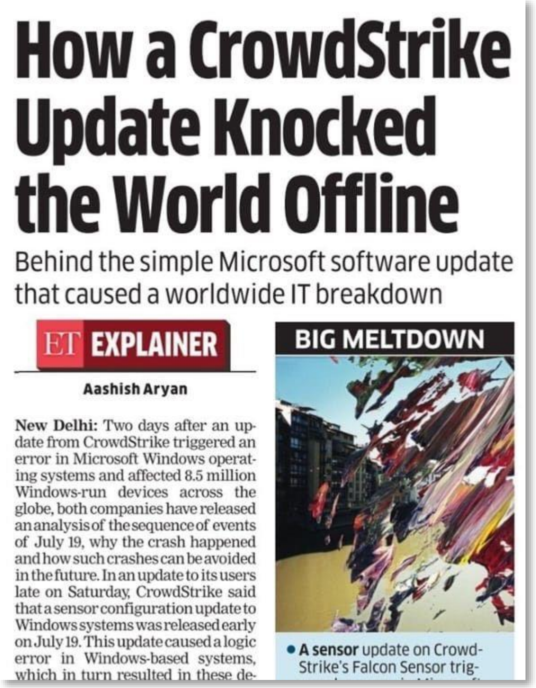
Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete

For more information about this issue and possible fixes,

If you call a support person, give them this info:

Stop code: CRITICAL_PROCESS_DIED



How a CrowdStrike Update Knocked the World Offline

Behind the simple Microsoft software update that caused a worldwide IT breakdown

ET EXPLAINER

Aashish Aryan

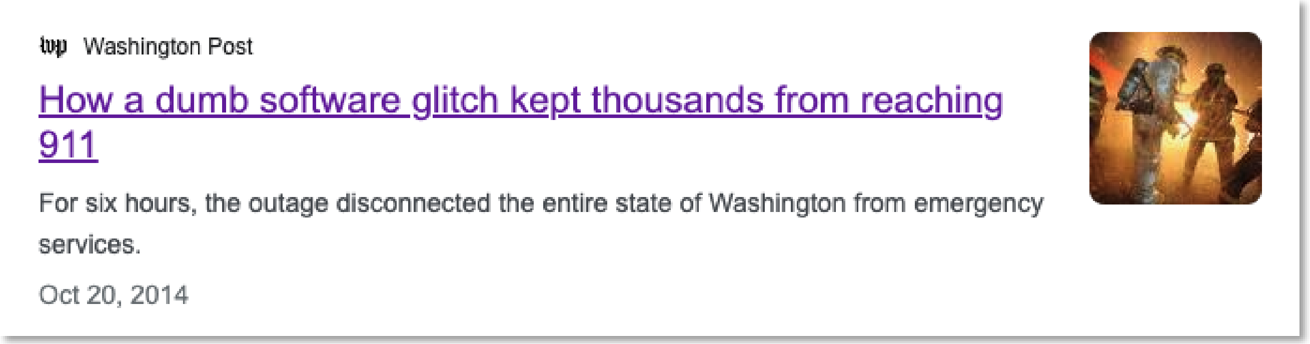
New Delhi: Two days after an update from CrowdStrike triggered an error in Microsoft Windows operating systems and affected 8.5 million Windows-run devices across the globe, both companies have released an analysis of the sequence of events of July 19, why the crash happened and how such crashes can be avoided in the future. In an update to its users late on Saturday, CrowdStrike said that a sensor configuration update to Windows systems was released early on July 19. This update caused a logic error in Windows-based systems, which in turn resulted in these de-

BIG MELTDOWN

- **A sensor** update on Crowd-Strike's Falcon Sensor trig-

911 Outage on April 2014

- [April 10, 2014] Washington State had **no 911 service for six hours**
  - 911 is the phone number for emergency service in USA
  - People calling were getting the busy signal
    - A woman called more than 37 times while a stranger was breaking into her house
  - People at the central office were not aware of the problem
  - More at: https://www.theatlantic.com/technology/archive/2017/09/saving-the-world-from-code/540393/



wp Washington Post

How a dumb software glitch kept thousands from reaching 911

For six hours, the outage disconnected the entire state of Washington from emergency services.

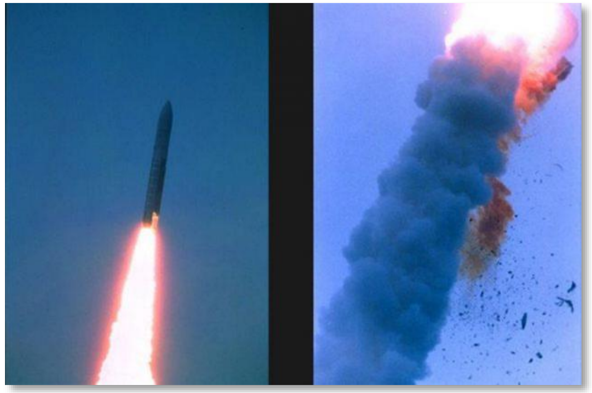
Oct 20, 2014

911 Outage on April 2014

- Why did this happen? **Software issue!**
    - The software dispatching the calls had a counter used to assign a unique identifier to each call
    - The counter went over the threshold defined by developers…
    - All calls from that moment on were rejected

More on failures… Ariane 5, 1996

- [June 4, 1996] 40s after take off, Ariane 5 broke up and exploded
  - The total cost for developing the launcher has been of 8000M$
  - The launcher contained a cluster of satellites for a value of 500M$
  - More at
    - http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html (accident tech report)
    - https://www.youtube.com/watch?v=PK_yguLapgA (video)



More on failures… Ariane 5, 1996

- The explosion has been caused by a software failure
- From the tech report:
  - “The failure […] was caused by the complete loss of guidance and altitude information […]. This loss of information was due to **specification and design errors** in the software of the inertial reference system.”
  - “The extensive reviews and tests carried out during the Ariane 5 Development Programme **did not include adequate analysis and testing** of the inertial reference system or of the complete flight control system, which could have detected the potential failure.”

Software engineering: definition

- Field of computer science dealing with software systems
    - large and complex
    - built by teams
    - exist in many versions
    - last many years
    - undergo changes
- Multi-person construction of multi-version software

Software engineer: required skills

- **Programming skills not enough!**
  - Programmer:
    - develops a complete program
    - works on known specifications
    - works individually
  - Software engineer:
    - identifies requirements and develops specifications
    - designs a component to be combined with other components, developed, maintained, used by others; component can become part of several systems
    - collaborates with others to ensure components show adequate qualities
    - works in a team

Skills of software engineers

- Technical
- Project management
- Cognitive
- Enterprise organization
- Interaction with different cultures
- Domain knowledge

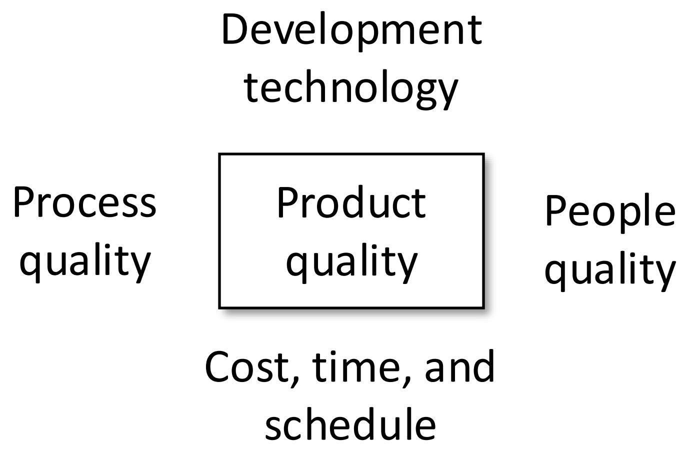
- The quality of human resources is of primary importance

Process and product

- Our goal is to develop **software products**
- The **process** is how we do it
- Both are extremely important, due to the nature of the software product
- Both have qualities
  - in addition, quality of process affects quality of product
  - …even though, other aspects such as the quality of the development team are important as well

The software product

- Different from traditional types of products
  - intangible
    - difficult to describe and evaluate
  - malleable
  - human intensive
    - does not involve any trivial manufacturing process
- Aspects affecting product quality



Development technology

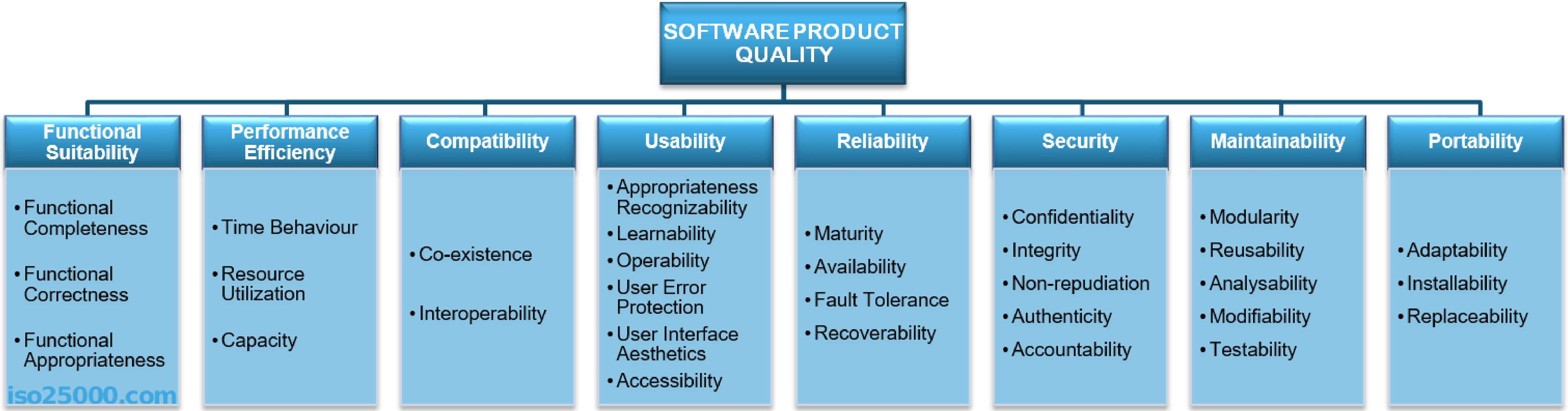
Process quality

Product quality

People quality

Cost, time, and schedule

Software product qualities – ISO/IEC 25010:2011 Software Quality Model



SOFTWARE PRODUCT QUALITY

Functional Suitability
- Functional Completeness
- Functional Correctness
- Functional Appropriateness

Performance Efficiency
- Time Behaviour
- Resource Utilization
- Capacity

Compatibility
- Co-existence
- Interoperability

Usability
- Appropriateness Recognizability
- Learnability
- Operability
- User Error Protection
- User Interface Aesthetics
- Accessibility

Reliability
- Maturity
- Availability
- Fault Tolerance
- Recoverability

Security
- Confidentiality
- Integrity
- Non-repudiation
- Authenticity
- Accountability

Maintainability
- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

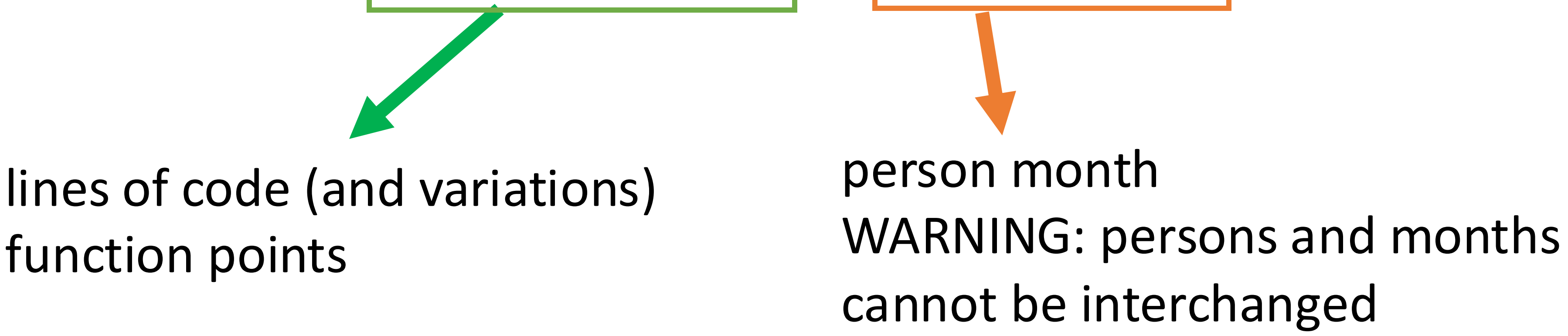
Portability
- Adaptability
- Installability
- Replaceability

iso25000.com

https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Process qualities: productivity

- Ability to produce a “good” amount of product
- How can we measure it? → Delivered items by unit of effort

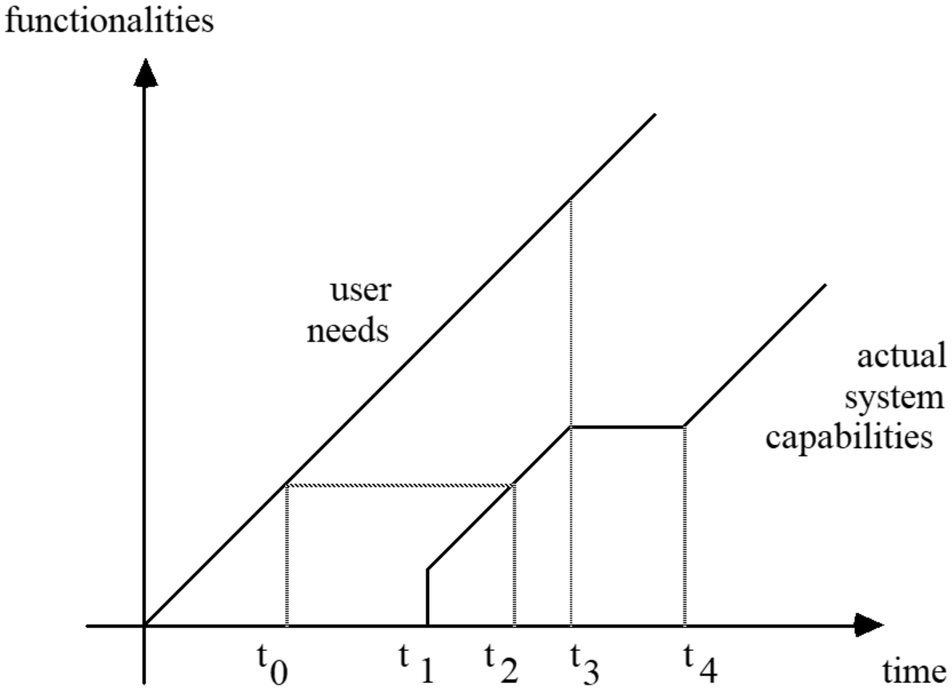


lines of code (and variations)
function points

person month
WARNING: persons and months cannot be interchanged

Process qualities: timeliness

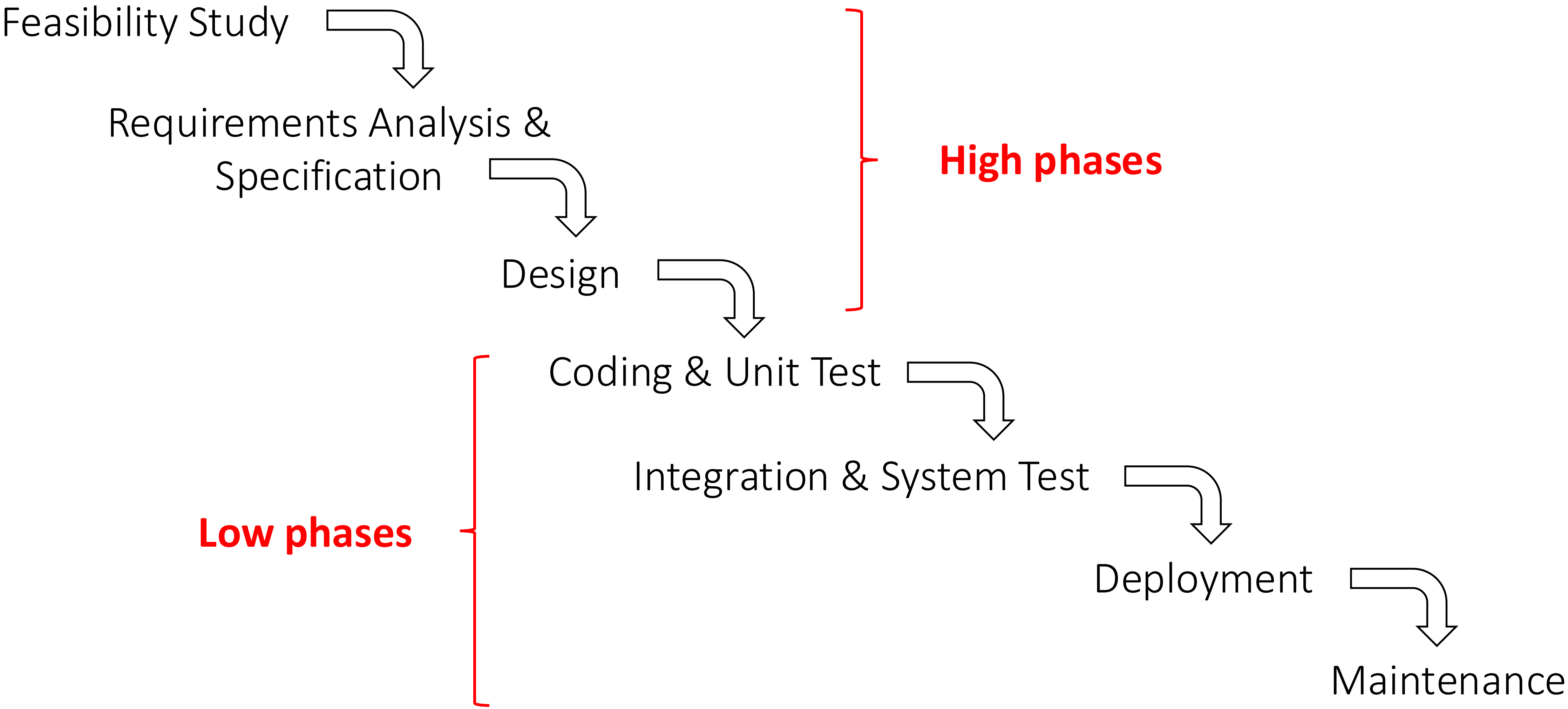
- Ability to respond to change requests in a timely fashion



Software lifecycles

- Initially, no reference model: Just code & fix
- As a reaction to the many problems: traditional “waterfall” model
  - identify phases and activities
  - force linear progression from a phase to the next
  - no returns (they are harmful)
    - better planning and control
  - standardize outputs (artifacts) from each phase
  - Software like manufacturing
- Then, flexible processes: iterative models, agile methods, DevOps

A waterfall organization



Feasibility Study

Requirements Analysis & Specification

Design

Coding & Unit Test

Integration & System Test

Deployment

Maintenance

High phases

Low phases

Feasibility study & project estimation

- Cost/benefit analysis
- Determines whether the project should be started (e.g., buy vs make), possible alternatives, needed resources
- Outcome
    - **Feasibility Study Document**
        - Preliminary problem description
        - Scenarios describing possible solutions
        - Costs and schedule for the different alternatives

Requirement analysis and specification

- Analyze the domain in which the application takes place
- Identify requirements
- Derive specifications for the software
  - Requires an (continuous) interaction with the user
  - Requires an understanding of the properties of the domain
- Outcome
  - **Requirements Analysis and Specification Document (RASD)**

Design

- Defines the software architecture
  - Components (modules)
  - Relations among components
  - Interactions among components
- Goal
  - Support concurrent development, separate responsibilities
- Outcome
  - **Design Document**

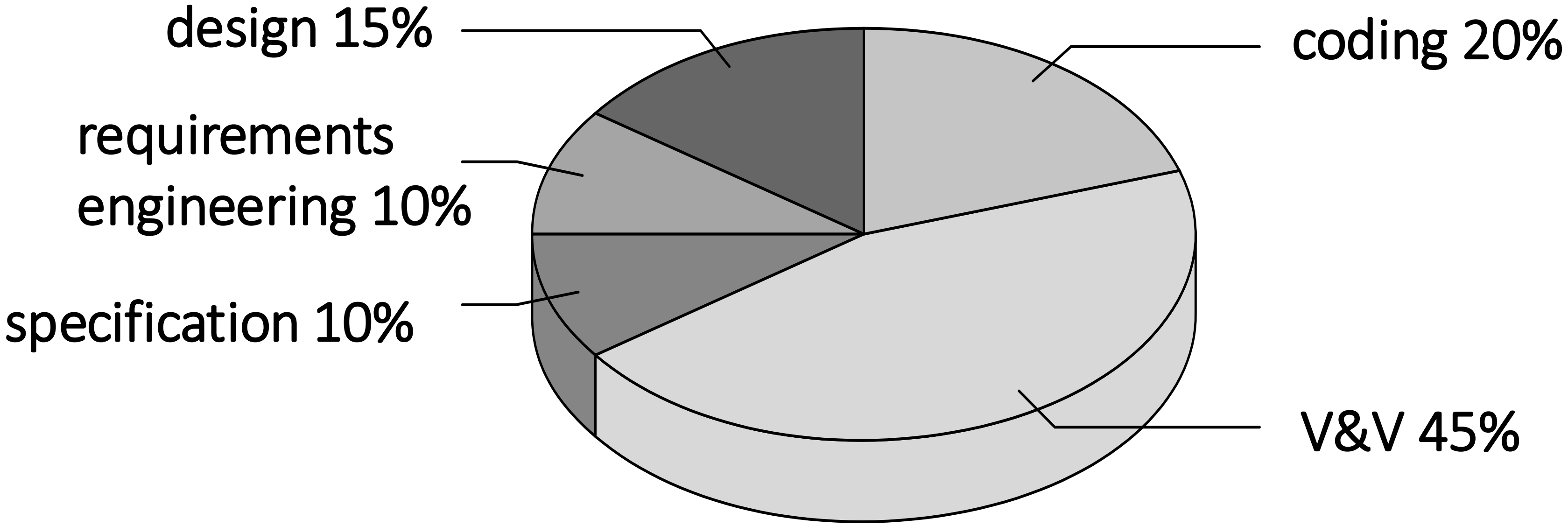
Coding & unit-level quality assurance

- Each module is implemented using the chosen programming language
- Each module is tested in isolation by the module developer
- Inspection can be used as an additional quality assurance approach
- Programs include their documentation

Integration & System testing

- Modules are integrated into (sub)systems
- Integrated (sub)systems are tested
- Follows an incremental implementation scheme
- Complete system test needed to verify overall properties
- Sometimes we have **alpha test** and **beta test**

Effort distribution (van Vliet 2008)



Maintenance

- Corrective maintenance: deals with the repair of faults or defects found ≈ 20%
- Evolution
  - **adaptive** maintenance: consists of adapting software to changes in the environment (the hardware or the operating system, business rules, government policies…) ≈ 20%
  - **perfective** maintenance: mainly deals with accommodating to new or changed user requirements ≈ 50%
  - **preventive** maintenance: concerns activities aimed at increasing the system’s maintainability ≈ 10%

Correction vs evolution

- Distinction can be unclear, because specifications are often incomplete and ambiguous
- This causes problems because specs are often part of a contract between developer and customer
  - early frozen specs can be problematic, because they are more likely to be wrong

Problems of software evolution

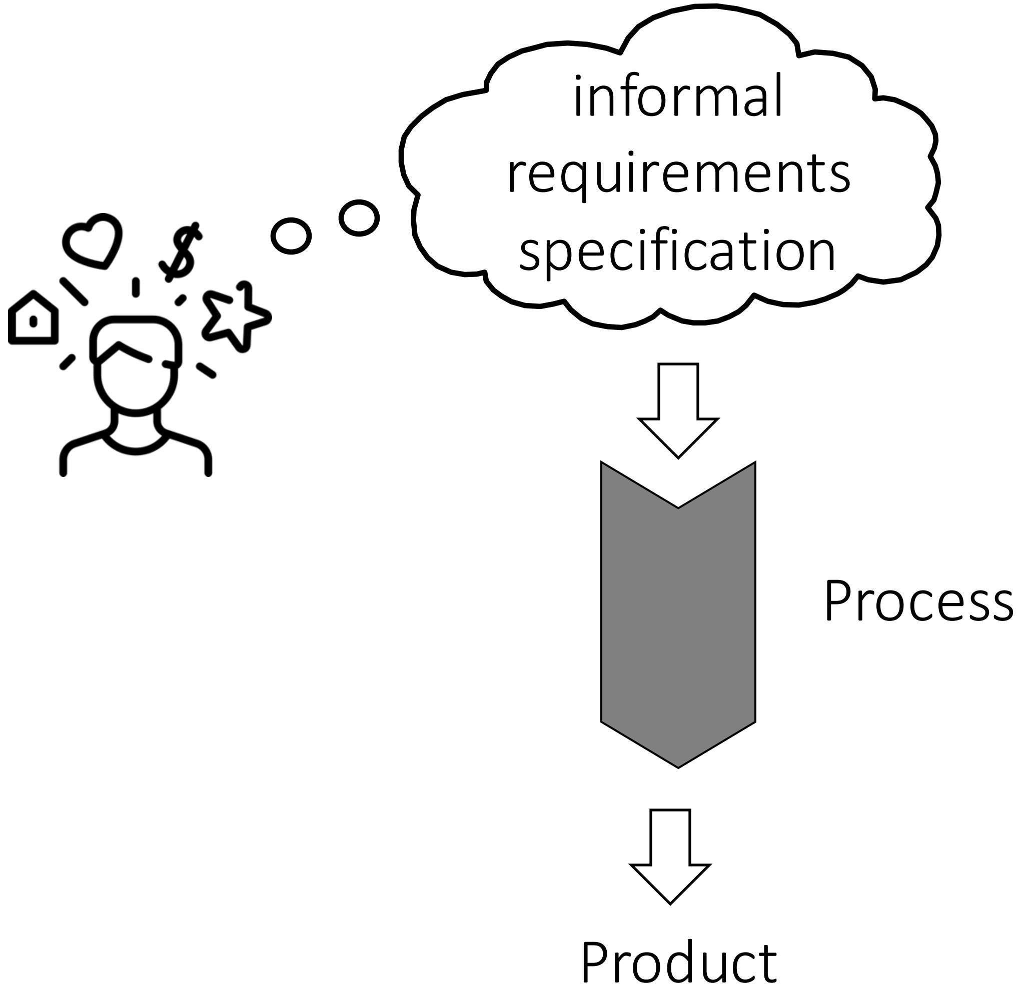
- It is (almost) never anticipated and planned; this causes disasters
- Software is very easy to change
  - often, under emergency, changes are applied directly to code
  - inconsistent state of project documents

How to face evolution

- Good engineering practice:
  - first modify design, then change implementation
  - apply changes consistently in all documents
- Likely changes must be anticipated
- Software must be designed to accommodate changes in a cost-effective way

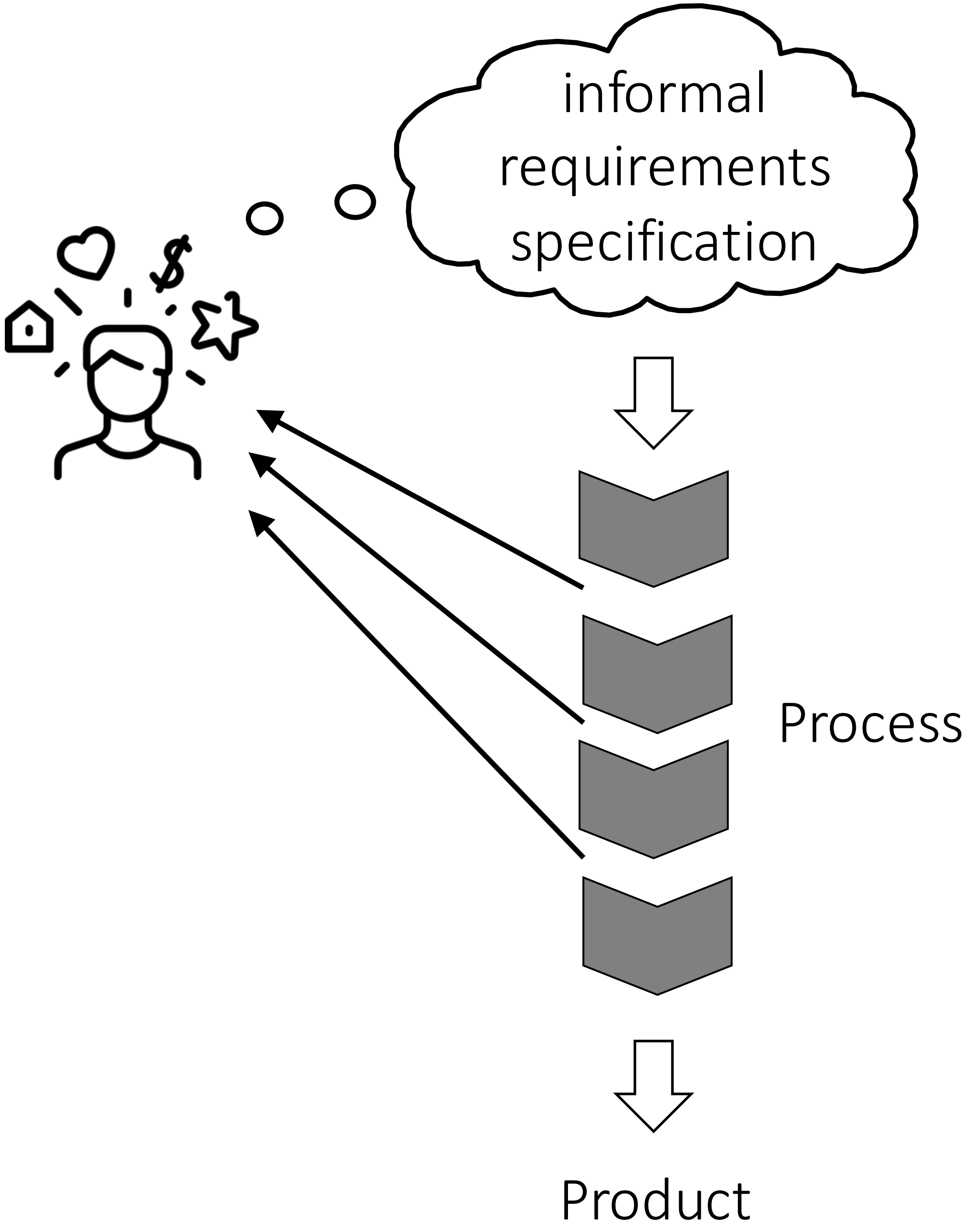
- **This is one of the main goals of software engineering**

Waterfall is “black box”

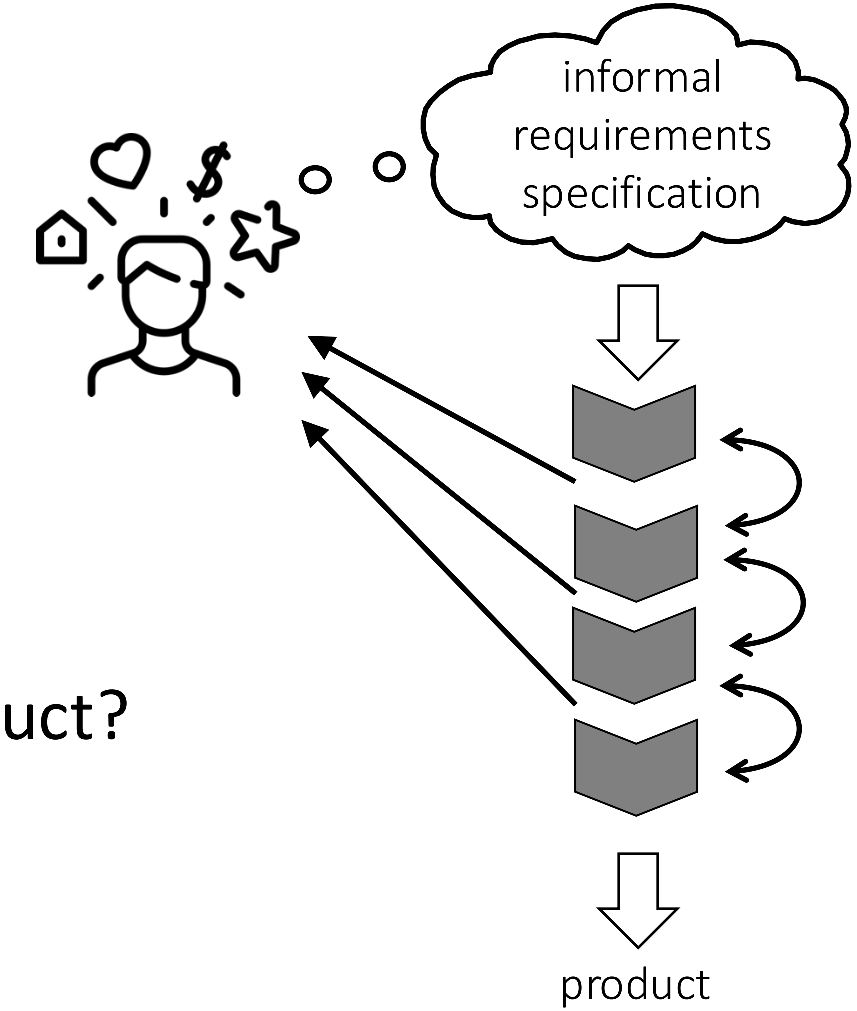


Need for transparency

- Transparency allows early check and change via feedback
- It supports flexibility



Verification and validation



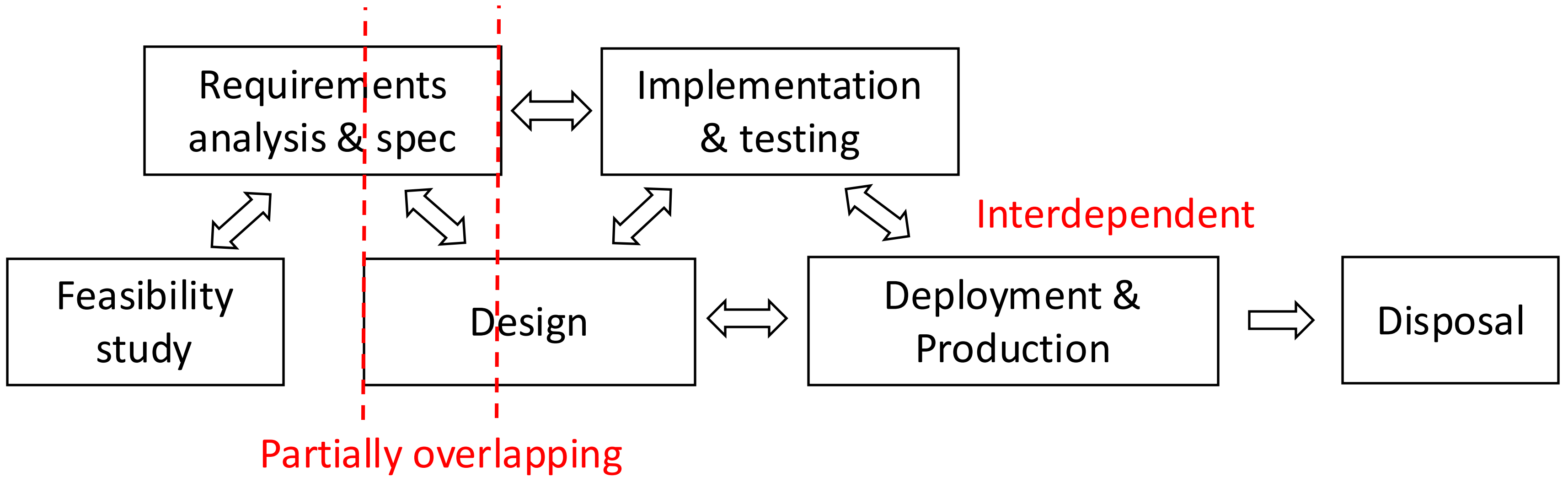
Validation:
are we making
the ”right” product?

Verification:
are we doing
the product right?

Flexible processes

- Main goal: adapt to changes (especially in requirements and specs)
- The very idea:
  - stages are not necessarily sequential
  - processes become iterative and incremental

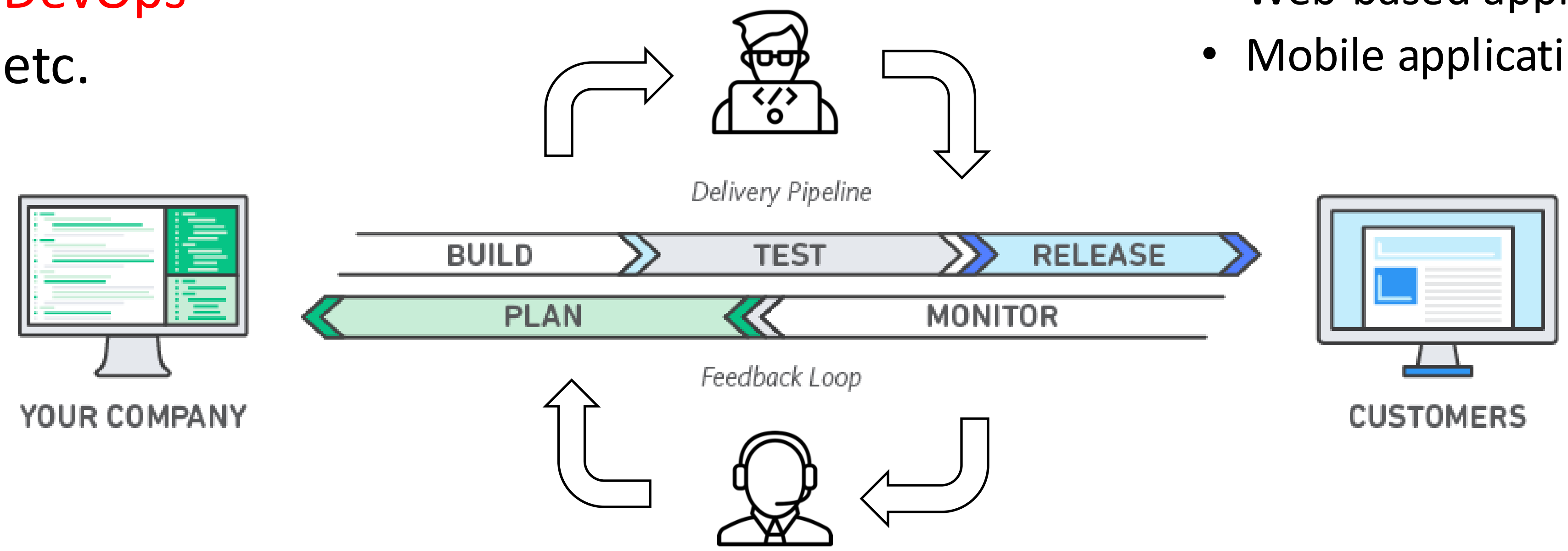
Example



Flexible processes

- Exist in many forms
  - eXtreme Programming (XP)
  - SCRUM
  - DevOps
  - etc.

- Effective in dynamic contexts
  - Many changes per week
  - Example:
    - Web-based applications
    - Mobile applications


Delivery Pipeline
BUILD
TEST
RELEASE
PLAN
MONITOR
Feedback Loop
YOUR COMPANY
CUSTOMERS