ENGINEERING & LIFECYCLES

SOFTWARE ENGINEERING

- 1958 – John Turkey first uses the term “software” in the sense used today.
- 1968 – NATO Conference on Software Engineering and the “Software Crisis”
- 1970 – Winston Royce publishes the (in)famous “waterfall” process paper – the foundations for years of rigid, slow-moving software development processes
- 1972 – IEEE Publishes first Transactions on Software Engineering
- 1976 – IEEE Computer Society established first committee for standards in software engineering
- 2001 – Agile Manifesto – The pendulum has swung all the way.
- Now – We are beginning to see a return to engineering principles and more process...

ARIANE 5 - 1996

What happened?

- Code reusing from Ariane 4
- Some of that code did not have a function in Ariane 5, but was kept nonetheless
- That code translated a 64bit value into a signed 16bit integer in an unsafe way
    - Never a problem for Ariane 4 – values physically limited to much lower values
    - But for Ariane 5 an overflow occurred and caused an exception
    - ...which was not caught and brought the primary system down.
    - ...the secondary backup system had the same software faults.

39 seconds into the flight, Ariane 5 self-destructed. It costed ~$370 million.

TSB INCIDENT – 2018

- 20 April – TSB begins a long planned IT upgrade:
  - Transfer around 5.2 million customer accoutns and records from old Lloyds Banking system to new bespoke TSB system.
  - Warns customers some services will not be available until 6pm on 22 April.
- 22 April – The IT migration has not gone to plan:
  - Login failures (<50% login success)
  - Seeing incorrect balances
  - Having access to other people's accounts
- 24 April – Up to 1.9 million customers remain locked out of their accounts.
  - Customers are encouraged to contact telephone banking team – Wait times of more than an hour
  - Financial Conduct Authority and Prudential Regulation Authority got involved.
- 26 April – TSB bring in a team from IBM
  - TSB says it will waive a $10m in overdraft fees and pay extra interest on current accounts
- 29 April – IBM provide a first preliminary report with a damning assessment of TSB's software engineering approach
- 4 September – TSB CEO Paul Pester steps down
  - Overall direct known cost to the bank >$175 million
  - Cost to TSB customers: unknown

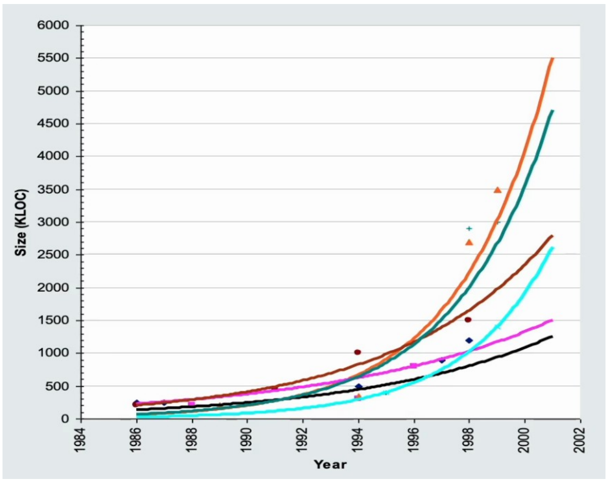
STATISTICS

Less than a third of software-development projects complete successfully!

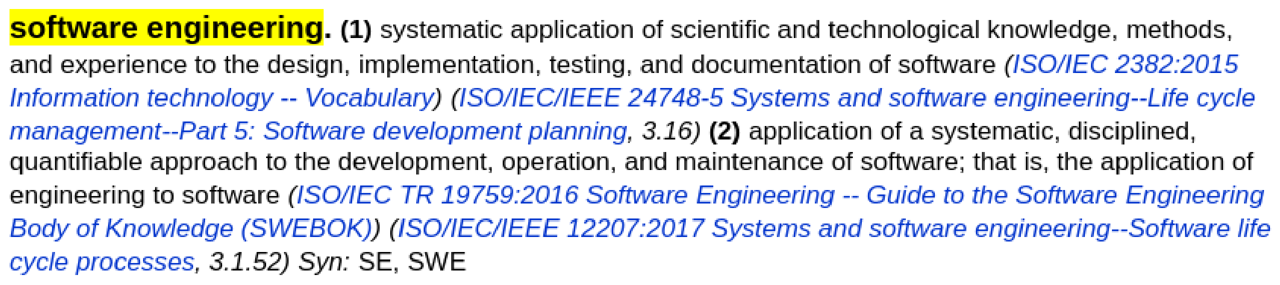
Where successful means on time, on budget and with a satisfactory outcome for stakeholders.

Around 50% are challenged, and the rest fail.

Our software grows increasingly more complex as time goes on. For example, every 5 years, Thales systems grow by a factor of approximately 5 to 10.



DEFINING SOFTWARE ENGINEERING



software engineering. (1) systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software *(ISO/IEC 2382:2015 Information technology -- Vocabulary) (ISO/IEC/IEEE 24748-5 Systems and software engineering--Life cycle management--Part 5: Software development planning, 3.16)* **(2)** application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software *(ISO/IEC TR 19759:2016 Software Engineering -- Guide to the Software Engineering Body of Knowledge (SWEBOK)) (ISO/IEC/IEEE 12207:2017 Systems and software engineering--Software life cycle processes, 3.1.52) Syn:* SE, SWE

www.computer.org/sevocab

ESSENTIAL ATTRIBUTES OF GOOD SOFTWARE

- Maintainability
  Software should be written in such a way so that it can **evolve** to meet the **changing needs of customers**. This is a critical attribute because software change is an inevitable requirement of a changing business environment
- Dependability and security
  Software dependability includes a range of characteristics, including **reliability, security, and safety.** Dependable software should **not cause physical or economic damage** in the event of a system failure. **Malicious users should not be able to access or damage the system.**
- Efficiency
  Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes **responsiveness, processing time, memory utilisation, etc.**
- Acceptability
  Software must be acceptable to the type of users for which it is designed. This means that it must be **understandable, usable, and compatible with other systems that they use.**

GENERAL ISSUES THAT AFFECT SOFTWARE

Heterogeneity

- Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices

Business and social change

- Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.

Security and trust

- As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

Scale

- Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.

SOFTWARE ENGINEERING DIVERSITY

There are many types of software systems and there is no universal set of software techniques that is applicable to all of these.

The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team.

APPLICATION TYPES

- Stand-alone applications
  These are application systems that **run on a local computer**, such as a PC. They include all necessary functionality and do not need to be connected to a network.
- Interactive transaction-based applications
  Applications that execute on a **remote computer** and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
- Embedded control systems
  These are software control systems that **control and manage hardware devices**. Numerically, there are probably more embedded systems than any other type of system.
- Batch processing systems
  These are business systems that are designed to **process data** in large batches. They process large numbers of individual inputs to create corresponding outputs.
- Entertainment systems
  These are systems that are primarily for **personal** use and which are intended to entertain the user.

- Systems for modelling and simulation
  These are systems that are developed by scientists and engineers to **model physical processes or situations**, which may include many, separate, interacting objects.
- Data collection systems
  These are systems that **collect data** from their environment using a set of sensors and send that data to other systems for processing.
- Systems of systems
  These are systems that are **composed** of a number of other software systems.

SOFTWARE ENGINEERING FUNDAMENTALS

Some fundamental principles apply to all types of software system, irrespective of the development techniques used:

- Systems should be developed using a **managed** and **understood** development process. Of course, **different processes are used for different types of software.**
- Dependability and performance are important for all types of system.
- Understanding and managing the software specification and requirements (what the software should do) are important.
- Where appropriate, **you should reuse software** that has already been developed rather than write new software.

INTERNET SOFTWARE ENGINEERING

- The web is now a platform for running application and organisations are increasingly developing web-based systems rather than local systems.
- Web services allow application functionality to be accessed over the web.
- Cloud computing is an approach to the provision of computer services where applications run remotely on the “cloud”.
  - Users do not buy software but pay according to use.

WEB-BASED SOFTWARE ENGINEERING

Web-based systems are **complex distributed systems** but the fundamental principles of software engineering discussed previously are as applicable to them as they are to any other types of system.

The fundamental ideas of software engineering apply to web-based software in the same way that they apply to other types of software system.

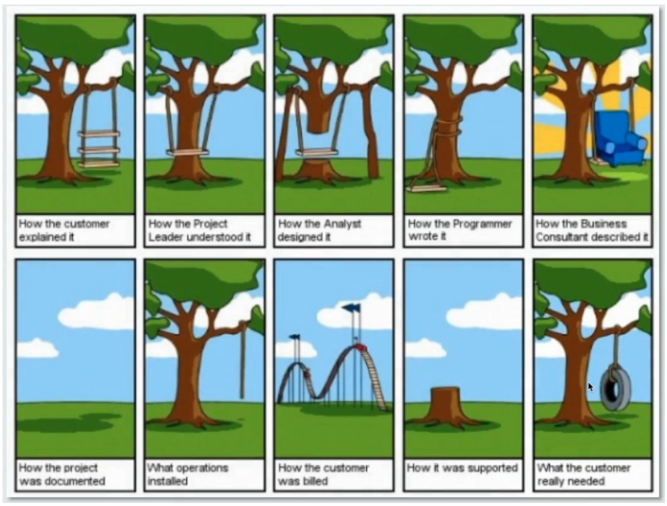
SOFTWARE DEVELOPMENT CYCLE

SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

Feasability Analysis → Requirements Analysis → Architecture & Design → Implementation → Testing → Deplyment & Maintenance → Feasability Analysis…

COMMUNICATION IS KEY

Building software for other people requires good communication between clients and your team.


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

FEASIBILITY ANALYSIS

Before you start on a project, get a sense of whether it can be done successfully.

Ask:

- Business need: Is there a business case for the system? Will it be used?
- Technical feasibility: Is it possible with available technology?
- Financial feasibility: Can it be developed within available budget?
- Time: Is it possible to develop in a useful time-frame?
- Resources: Are all necessary resources (people, tools) available for the development?

If a project proposal fails these tests, very high risk to attempt development!

REQUIREMENTS ANALYSIS

Systematically identifies and records requirements of stakeholders.

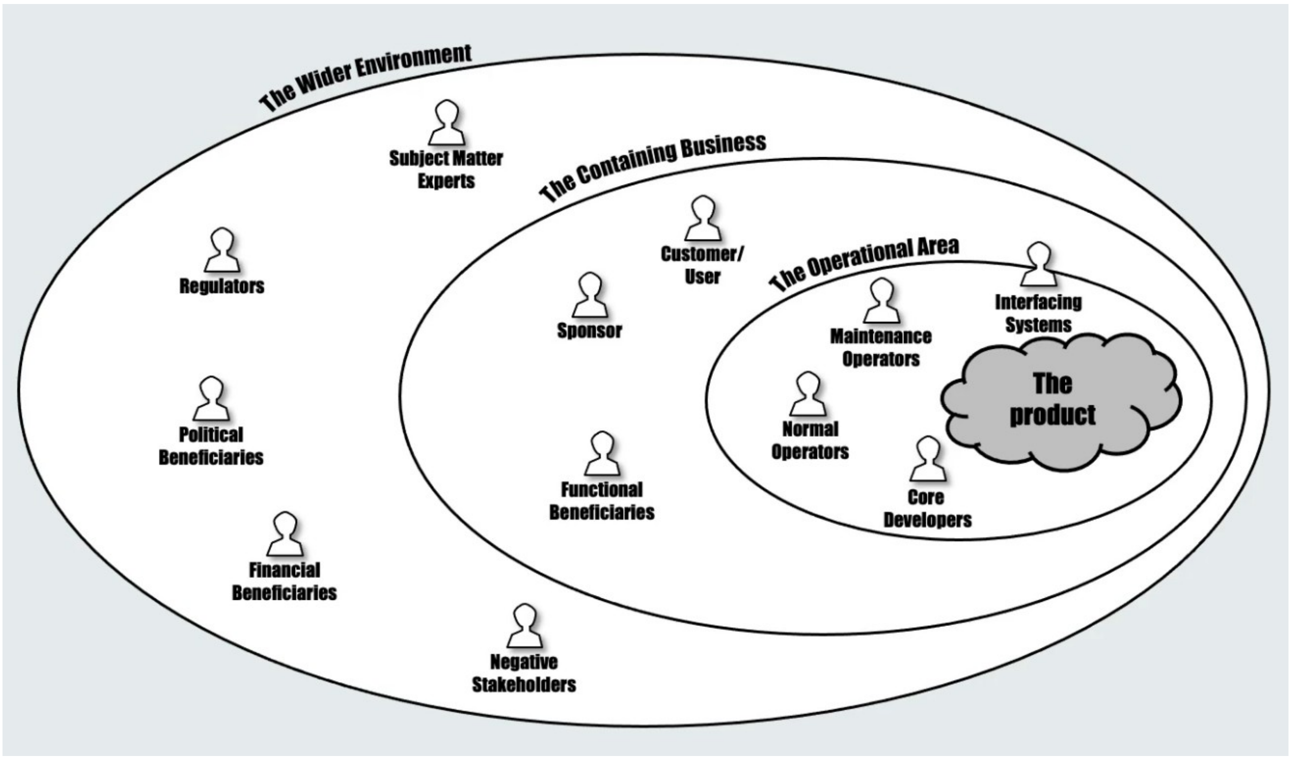
So what are:

- Stakeholders?
- Requirements?

Thorough requirements analysis can reduce errors and cost later in development

- But there's such a thing as analysing too much → "analysis paralysis"

STAKEHOLDERS



REQUIREMENTS

Requirements reflect the needs of stakeholders

- What should the system do (functional requirements)
- How the system should do it (non-functional or quality requirements, e.g., performance, usability, extensibility, ...)
- Constraints imposed by system context (e.g. existing systems, policy, law, ...)

Requirements may be conflicting or ambiguous

- Requirements engineering/analysis aims to resolve ambiguity and conflict.

Requirements should eventually be written up as a specification.

- Different forms: multi-page document, formal-logic specification, user stories, use cases, …
- Shouuld involve requirements prioritisation
  - e.g. following the MoSCoW style – Must, Should, Could, Won't
  - Based on risk/benefit analysis.
- Should be SMART – Specific, Measurable, Assignable, Realistic, Time-related.

STAGES IN REQUIREMENTS ANALYSIS

Domain Analysis, requirements elicitation

- Identify stakeholders
- Gather information on domain + requirements
- With users, customers, other stakeholders, literature, other similar systems, …

Evaluation and negotiation

- Identify conflicts, imprecision, commissions, redundancies
- Consult and negotiate with stakeholders to agree resolutions

Specification and documentation

- Systematically document requirements as system specification
- In precise, possibly formal, notation
- Agreement between developers and stakeholders on what will be delivered.

Validation and verification

- Check (formalised) requirements for consistency, completeness and correctness

REQUIREMENTS ANALYSIS TECHNIQUES

- Interviews with stakeholders
- Brainstorming sessions
- Observation of existing processes (ethnography)
- Scenario analysis – model specific scenarios of use of system, e.g. as UML sequence diagrams
- Document mining
- Goal decomposition
- Exploratory prototyping.

ARCHITECTURE

High-level, big picture view of the system

Good for work planning, overall team organisation and communication, information hiding, …

4+1 model:

- Logical: Define major subsystems (components) and their inter-connections
- Process: Define the major information and data-flows, their interactions and concurrency
- Physical: Identify hardware resources and allocation of components
- Development: Identify how to package, build (generate, compile, link) and deploy (install)
- Scenarios: Use cases highlighting integration/interaction between the views.

DESIGN

Thinking through detailed structure and algorithms *within* components:

Risk/Value driven:

- Invest amount of effort appropriate to each part of the system
- Don’t design everything, some things can simply be “programmed”
- Focus on parts that are technically difficult, risky, “known unknowns”

May involve diagrams in UML or other modelling languages

Enables high-level analysis

- For example, of performance, security, safety, reliability,

- possibly using automated tools

Produces sufficient understanding to start development

- May need to be signed off by relevant stakeholders.

IMPLEMENTATION

Building the software following the design documents

- Using appropriate languages and tools
- Involving relevant expertise from different areas
- Including code for physical and development views
- Infrastructure as code

TESTING

Aim is to discover errors before they make it into production

- White-box testing
- Black-box testing

Levels of testing

- Unit tests: test each component in isolation
- Integration tests: Test that components interact correctly
- System (end-to-end) testing: test entire system
- Acceptance tests (UATs) : test against requirements (black-box testing of use-case scenarios)

Regression testing

- Tests should be automated so that they can be re-run after each change
- Ensures no “regressions” occur; that is, errors are not introduced inadvertently.

DEPLOYMENT

Roll out software onto production system

- May require sophisticated support sy stems itself
- May need consideration of transition phase
- Roll out itself is potentially dangerous activity and may need substantial planning
- May require training of users

MAINTENANCE

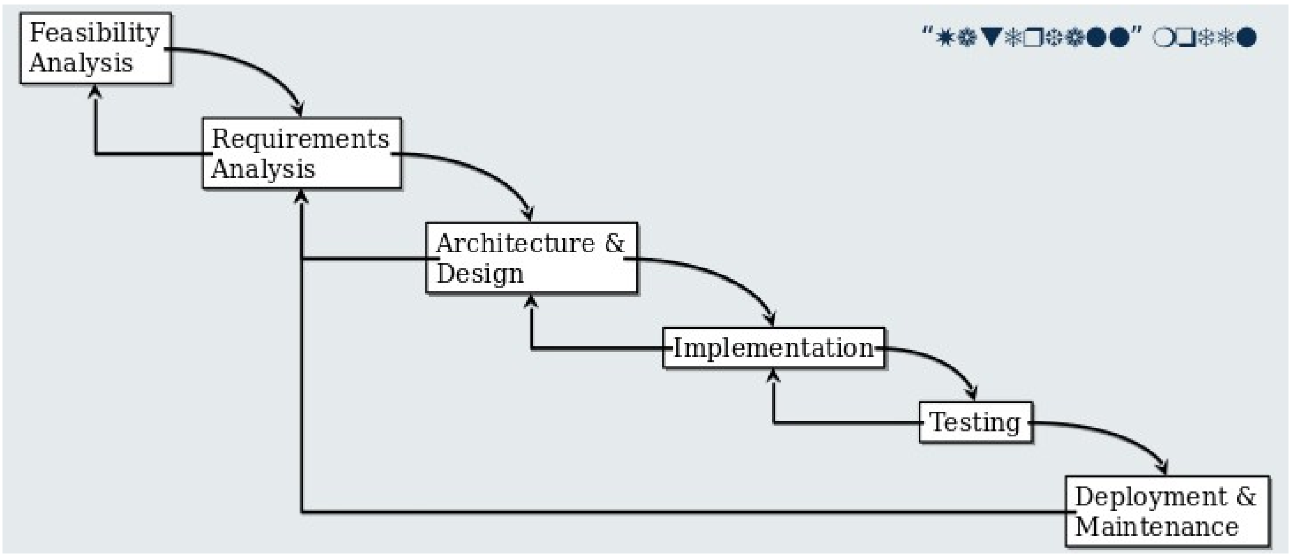
All post-delivery activities, including:

- Correction: bug-fixing and correcting defects
- Adaptation: changing system to operate in new/updated environment
- Enhancement: extension to handle new requirements
- Prevention: re-engineering to improve structure of system and enhance its future evolution
- Decommissioning: manage termination of use of system

These activities can consume far more resources than development of entirely new systems.

SELECTED SOFTWARE DEVELOPMENT LIFE CYCLES

WATERFALL METHOD



PROBLEMS WITH THE WATERFALL METHOD

What if the requirements change?

- Once captured, requirements are assumed to stay fixed.
- The requirements may have been misunderstood
- The requirements may change
  - “Wicked problems” → Stakeholders only understand their requirements when they see a (partial) solution.

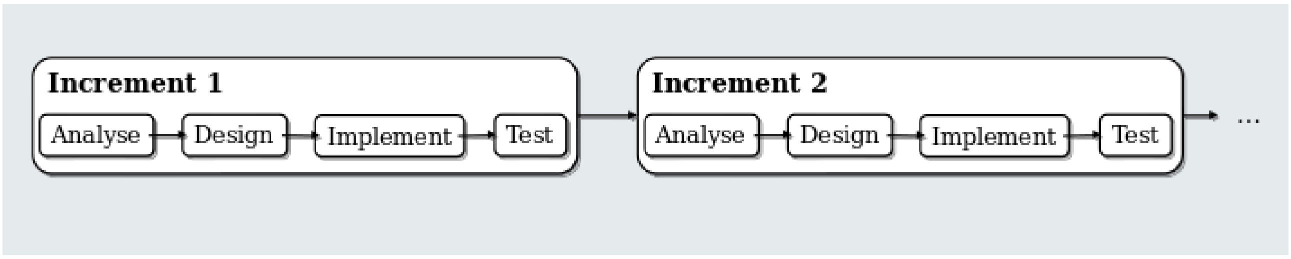
Time-to market and loss of system relevance

- System only available at end of process
- Might be a long time

Incorporation of risk/value in process?

- Always building complete system
- What if a step at the end fails?
- What if a component built at the start loses value for the customer?

INCREMENTAL (~AGILE) APPROACHES



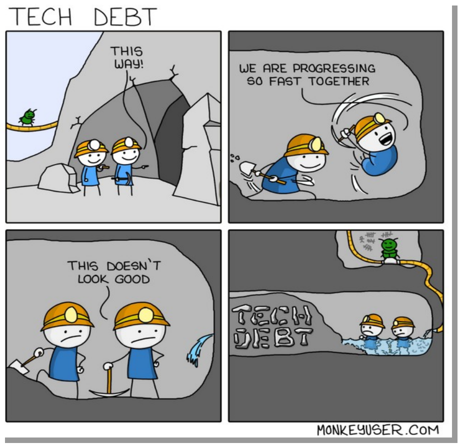
Design and plan increments

- Each increment provides additional functionality
- Planning horizon is across multiple increments
  - Can lay architectural/design foundations in earlier increments

PROS AND CONS

Pros

- Early value and continues to add value
- Flexibility to adjust to changing requirements
- Risk of failure reduced because increments are short
- Project always delivers something



Cons

- Difficult to identify end of project
- Difficult to plan costs and budgets
- Can easily deteriorate into code and fix
- Technical debt

AGILE CYCLES

Agile approaches favour incremental/iterative approaches

- Often driven by feature list
- Aka “Kanban board”, “product/spring backlog”,…
- Regularly viewed, reviewed and (re-)prioritised with product owner/relevant stakeholders

Planning horizon often very short-term

- Architectural concerns sometimes only really captured through Minimum Viable Product
- In SCRUM, sometimes through a “Sprint 0”, which does initial planning work.