Diploma in

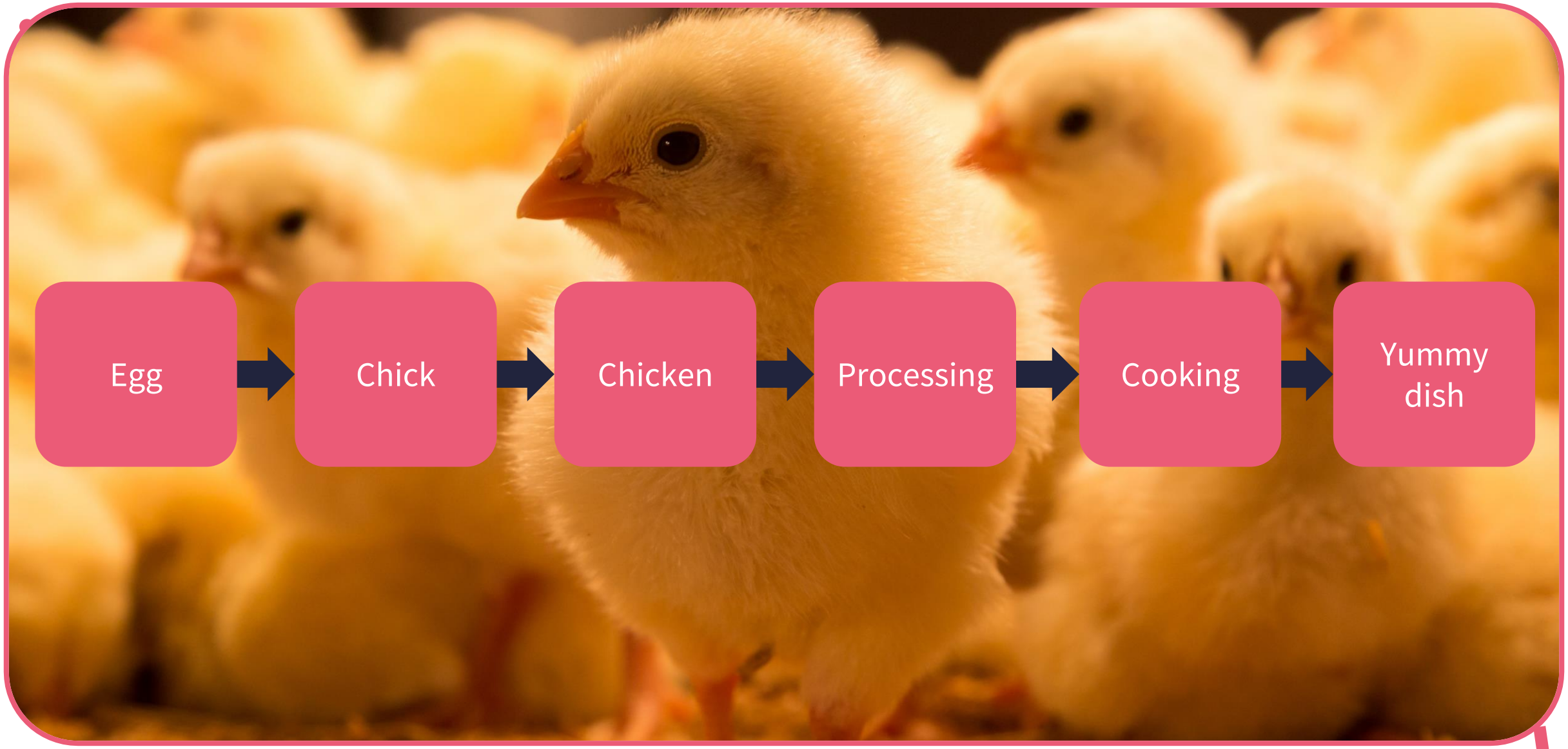
Computer Science

The Software Development Lifecycle

Objectives

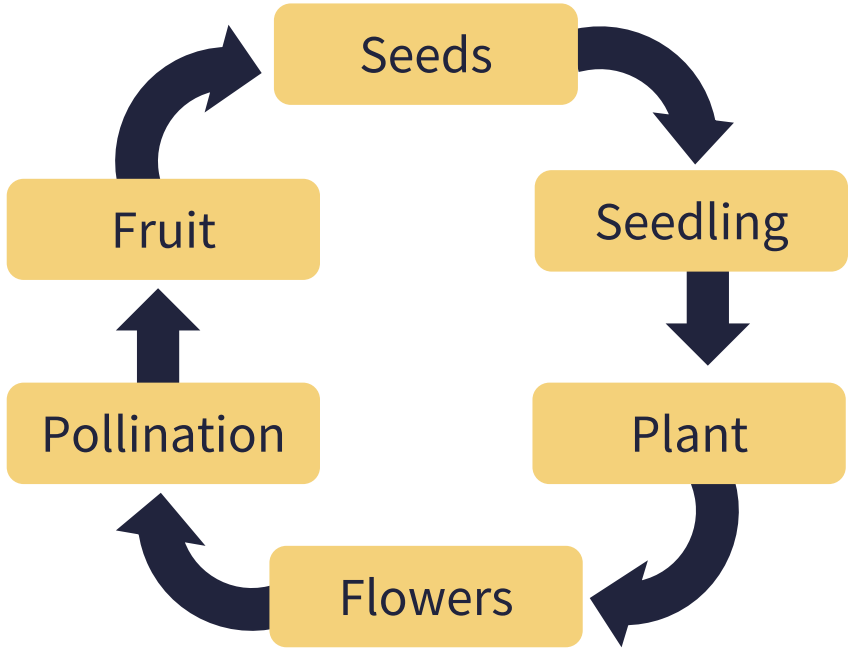
- Explain the strategies of software development
- Apply the software development life cycle in writing code
- Explore testing models
- Comprehend the pitfalls that can be encountered in software development and how to avoid them

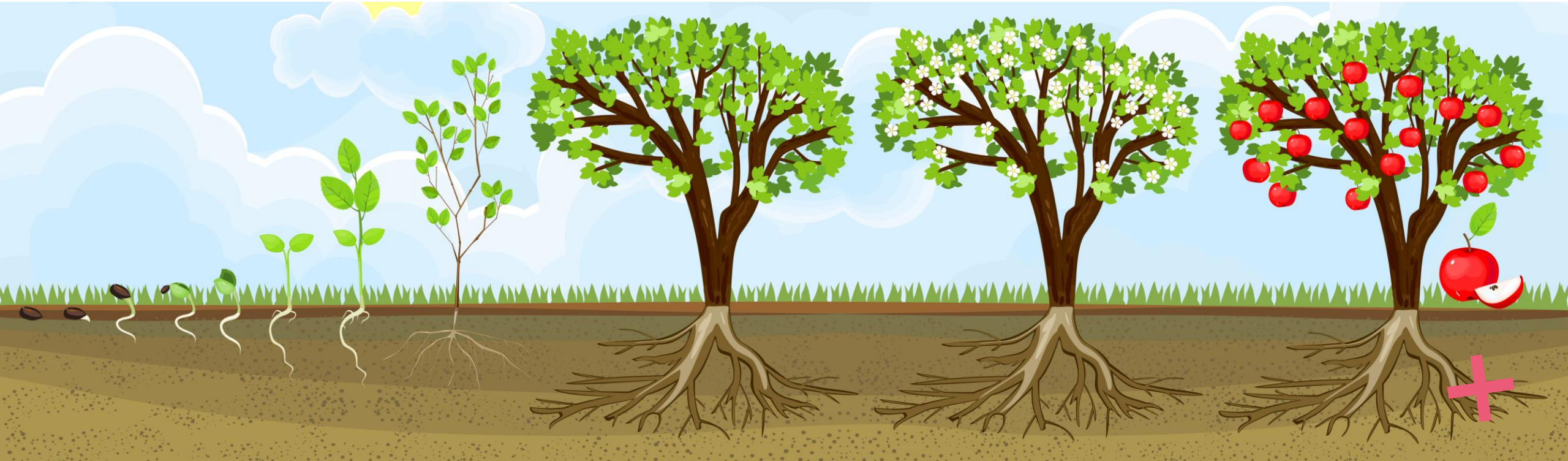
The software development process



Cycles in everyday life

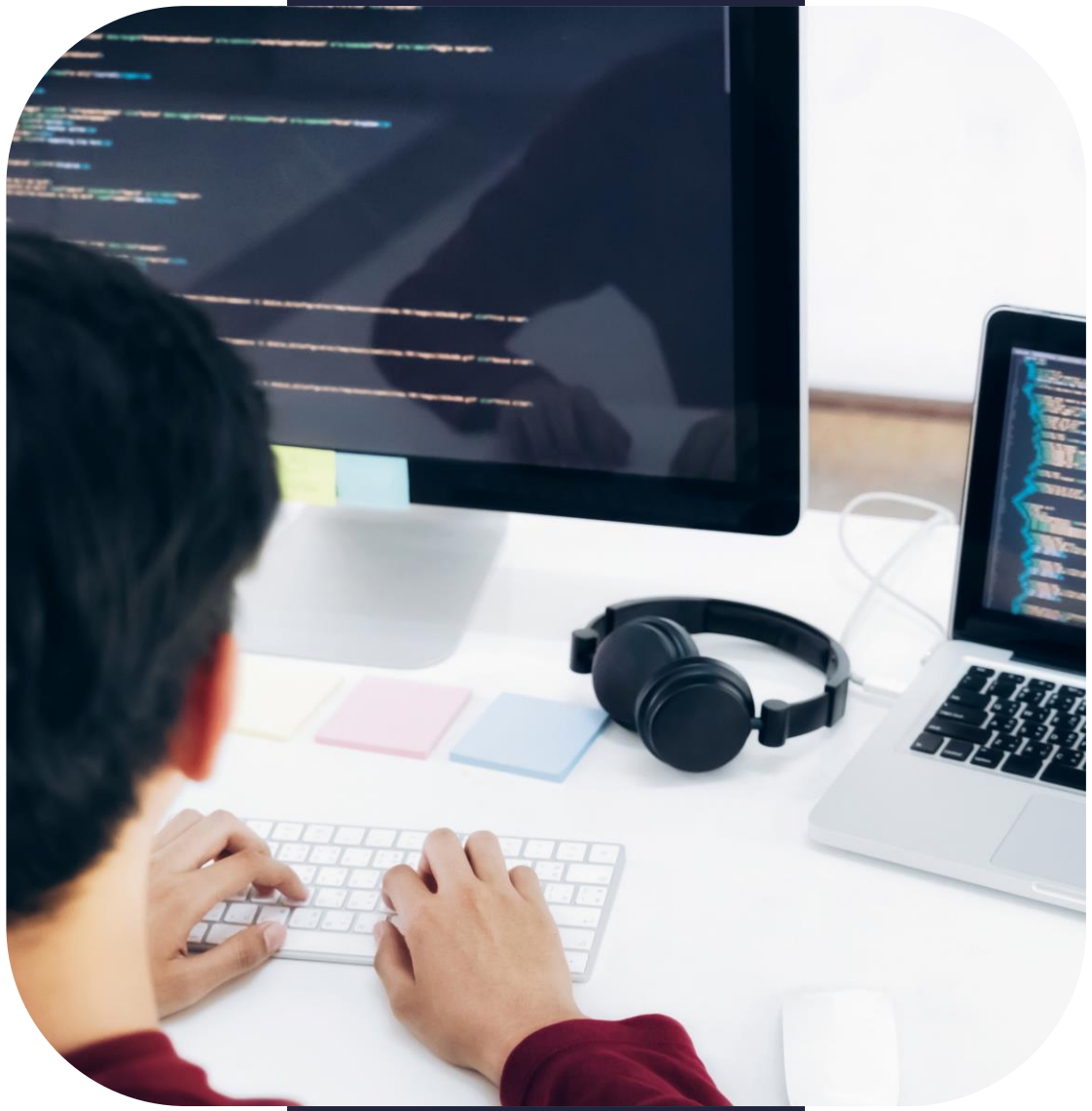
Cycles in everyday life





Idea → Problem statement → Software package → Decommission → Production maintenance & optimisation

Cycles in software



Software cycle

- Flow charts and pseudo code are vital
- Planning is key

Reasons for planning

Planning

- The more complex a project becomes, the more critical it is to plan
- Focuses your expertise on the right path

Reasons for planning

- Defining goals
- Determining requirements
- Determining costs
- Creating a timeline
- Improving software quality
- Understanding project’s value

Defining goals

- Helps develop the project
- Outline a realistic plan
- Helps you determine if the project is feasible

Determining requirements

- Direct and indirect
- Requirements will change with progress
- Need a solid plan to stay within budget and avoid failure

Determining costs

- Nobody wants to invest money without knowing the outcome
- Need to manage costs without sacrificing results

Creating a timeline

- Large software projects involve many people
- Project teamwork requires co-ordination
- Realistic targets must be set and milestones observed

Improving software quality

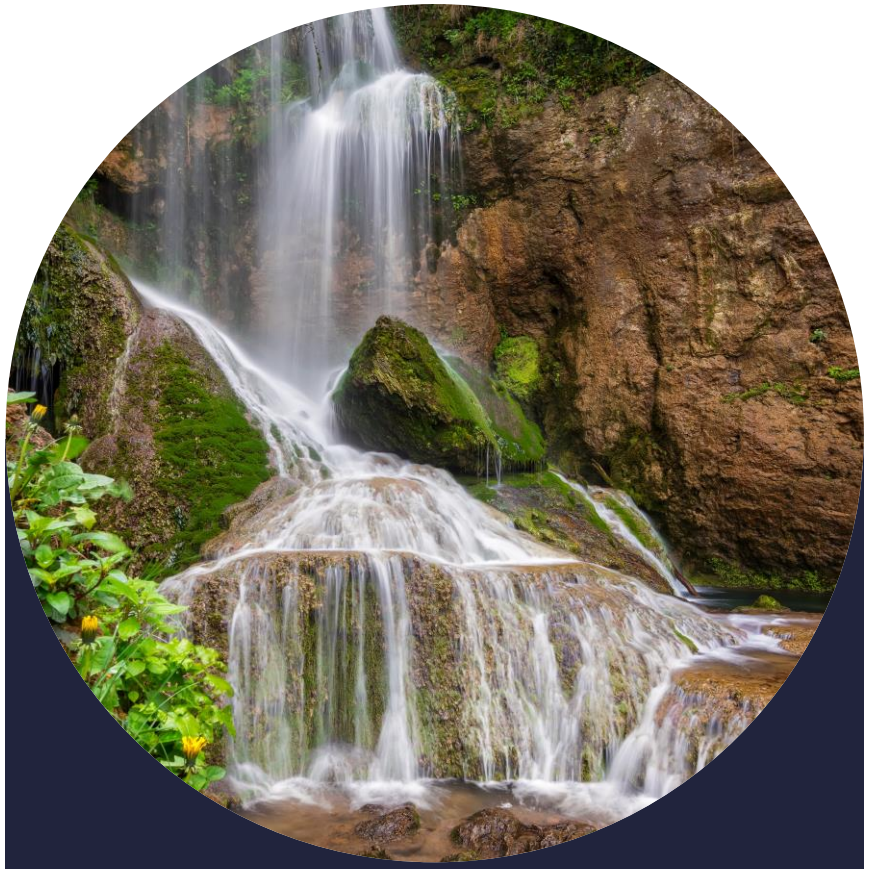
- Set boundaries
- Know when to stop
- Be better prepared for eventualities

Improving software quality

- Planning the project keeps you on top of things
- Planning also provides you with quality metrics
- Deviations need to be controlled

Understanding project's value

- Milestones and deliverables will provide a benchmark
- Need to manage your time and keep track of deadlines and costs



History of software development models

- First conceptual framework for structuring software development into phases was in 1970s - the waterfall model
- A linear model with milestones at various stages
- Has distinct phases – each with its own distinct activities
- V model in 1980s - an extension of the waterfall model

Newer software development models

- Customer-centricity showed flaws in the waterfall
- Changing requirements did not work well with waterfall model
- Iterative models appeared as agile frameworks – spiral model and RAD
- Other reworked models appeared in 1990s

Latest models

- Methods such as extreme programming developed to improve quality and cater for changing requirements
- Agile systems now in widespread use as rigid models can’t keep up

The software development life cycle (SDLC)

Stages in the SDLC

Analysis

Feasibility study

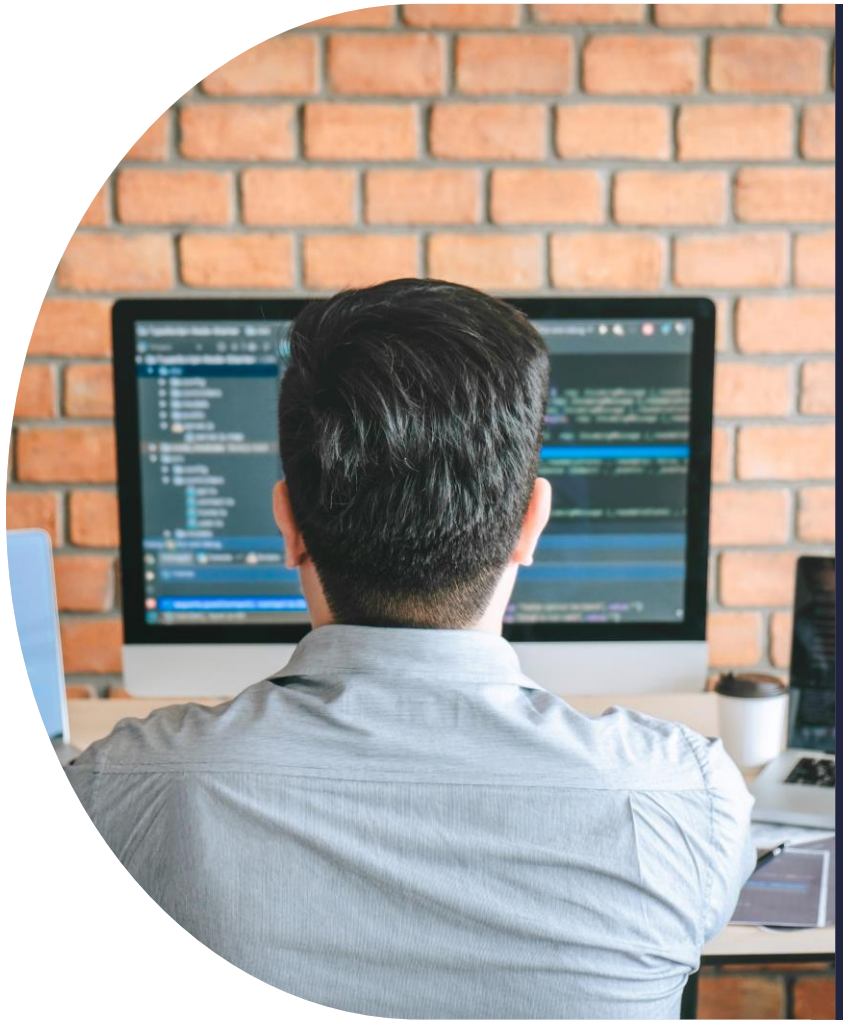
Design

Coding

Testing

Deployment

Maintenance



Analysis

- Is project a good idea?
- Requirements and vital info gathered from the user
- Objectives and user’s expectations are fleshed out
- Communication is vital

Analysis

Step 1: Requirements gathering

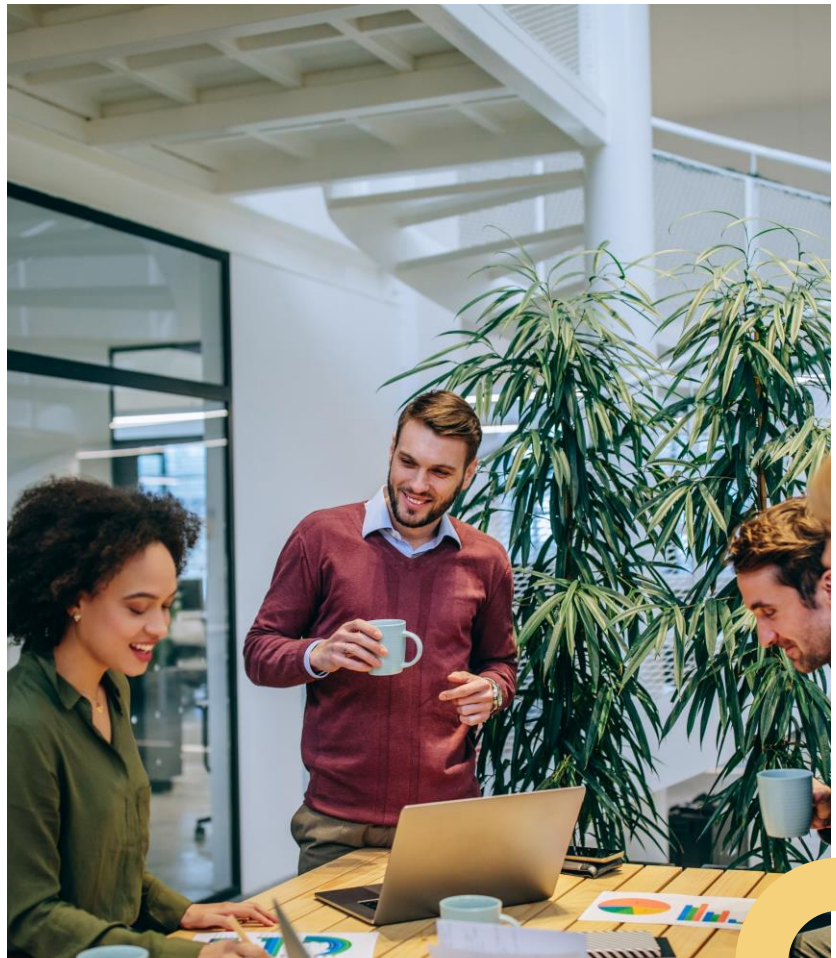
Step 2: Scope definition

Step 3: Planning

Analysis

- Thorough planning required to save time, money and resources
- Scope of the project is defined
- No single software can solve a set of problems entirely
- Scope sets limits while keeping client happy

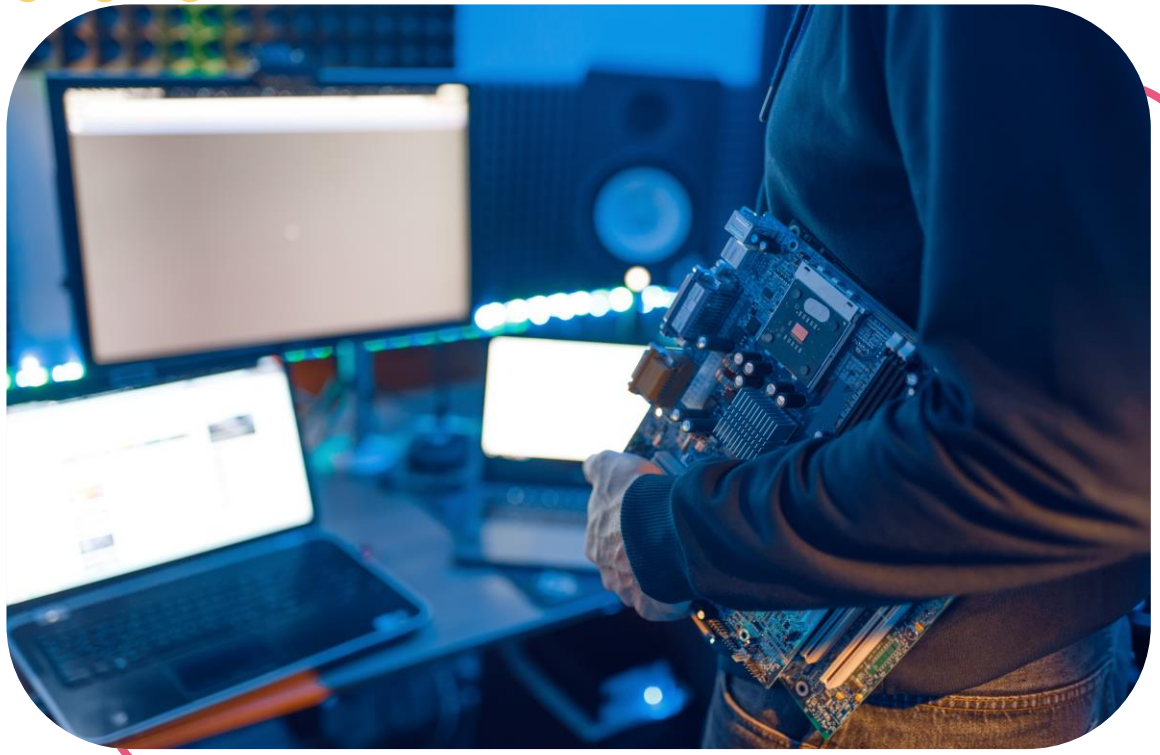
Feasibility study (& SRS document)



1. Technical
2. Economic
3. Legal
4. Operational
5. Schedule

Technical feasibility

- Can current computer systems handle the system we want to develop?
- What does the client have?

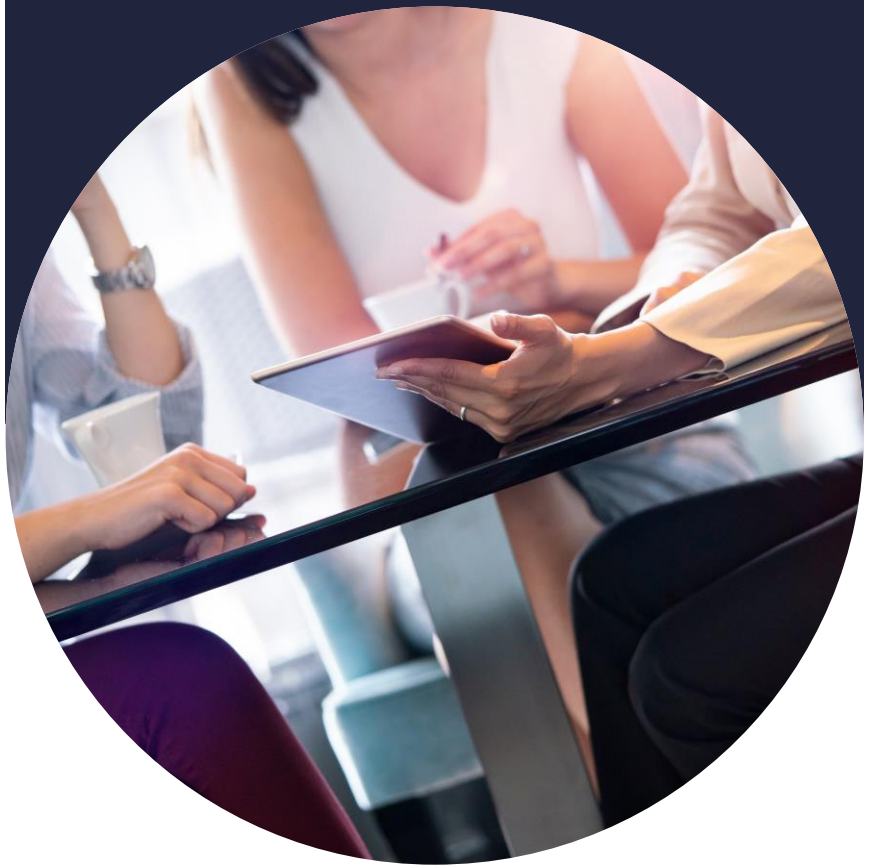


Economic feasibility

- Budget
- What can the client afford?
- What does the programmer need to earn?
- Goes hand-in-hand with the project scope

Legal feasibility

- Is the project legal?
- Easy to violate cyber laws and regulatory frameworks, compliance and policies

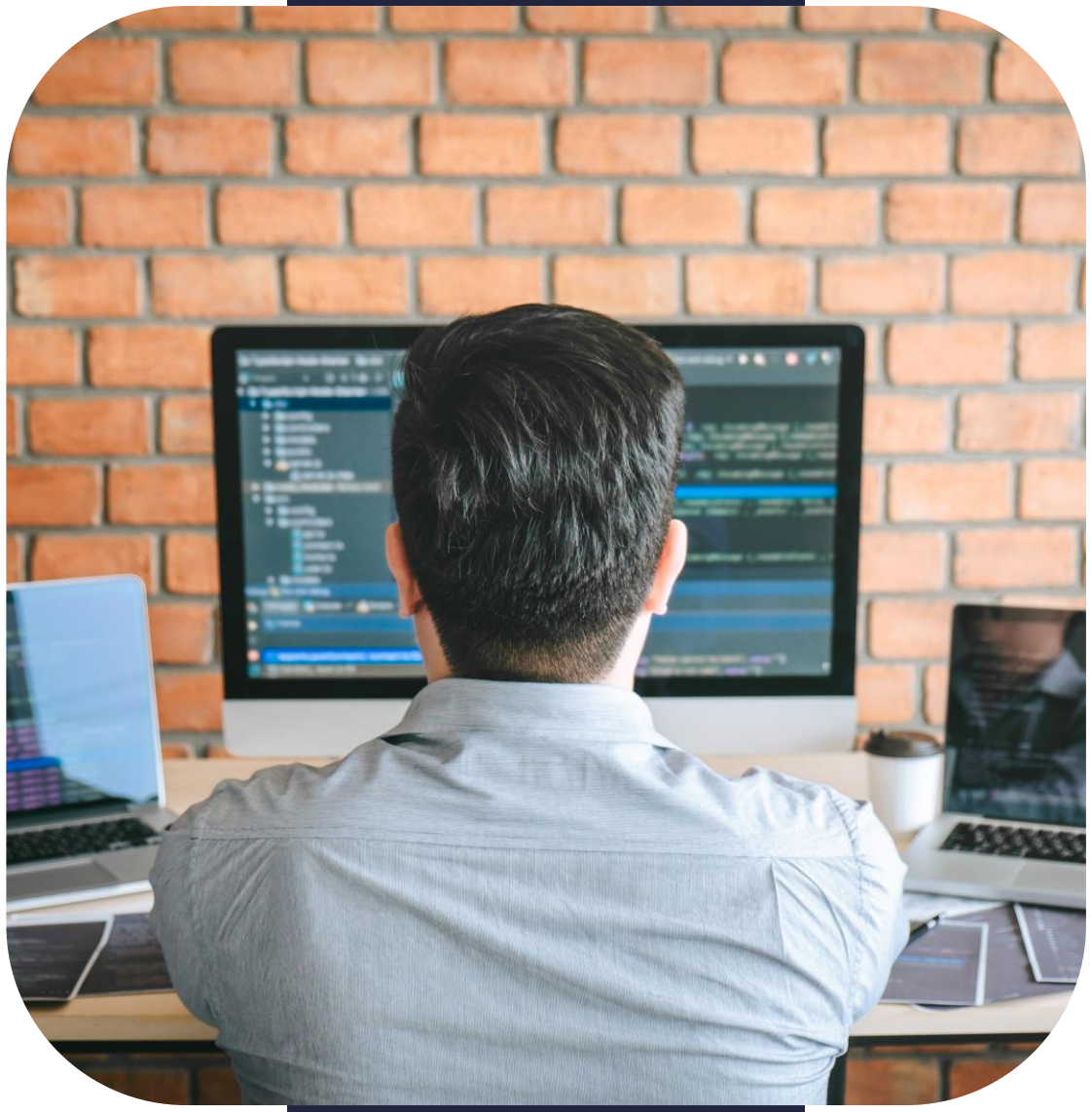


Operational feasibility

- Can the programmer create the operations and processes required?
- Looks at resources, skills, capabilities and degree of difficulty and maintenance

Schedule feasibility

- Looks at timelines and that each stage of development
- Allocating time while balancing quality and deliverables
- Crucial component – due to fast pace of software development industry



To summarise...

- The feasibility study depicts the direction in which the project will go at a high level.
- You are guaranteed to fail if you do not do a proper feasibility study!

Design

Two main documents:

- High-level design document
- Low-level design document

High-level design document

- Brief description of each module and proposed name
- Flowcharts and tables
- Functionality of every module is outlined

Low-level design document

- Contains functional logic
- User interface is detailed
- Error messages are documented
- Inputs and outputs paired with modules

Coding

- Developers write the actual code using design documents
- Longest phase of the life cycle
- Constant referencing to preceding documents and stages

Testing

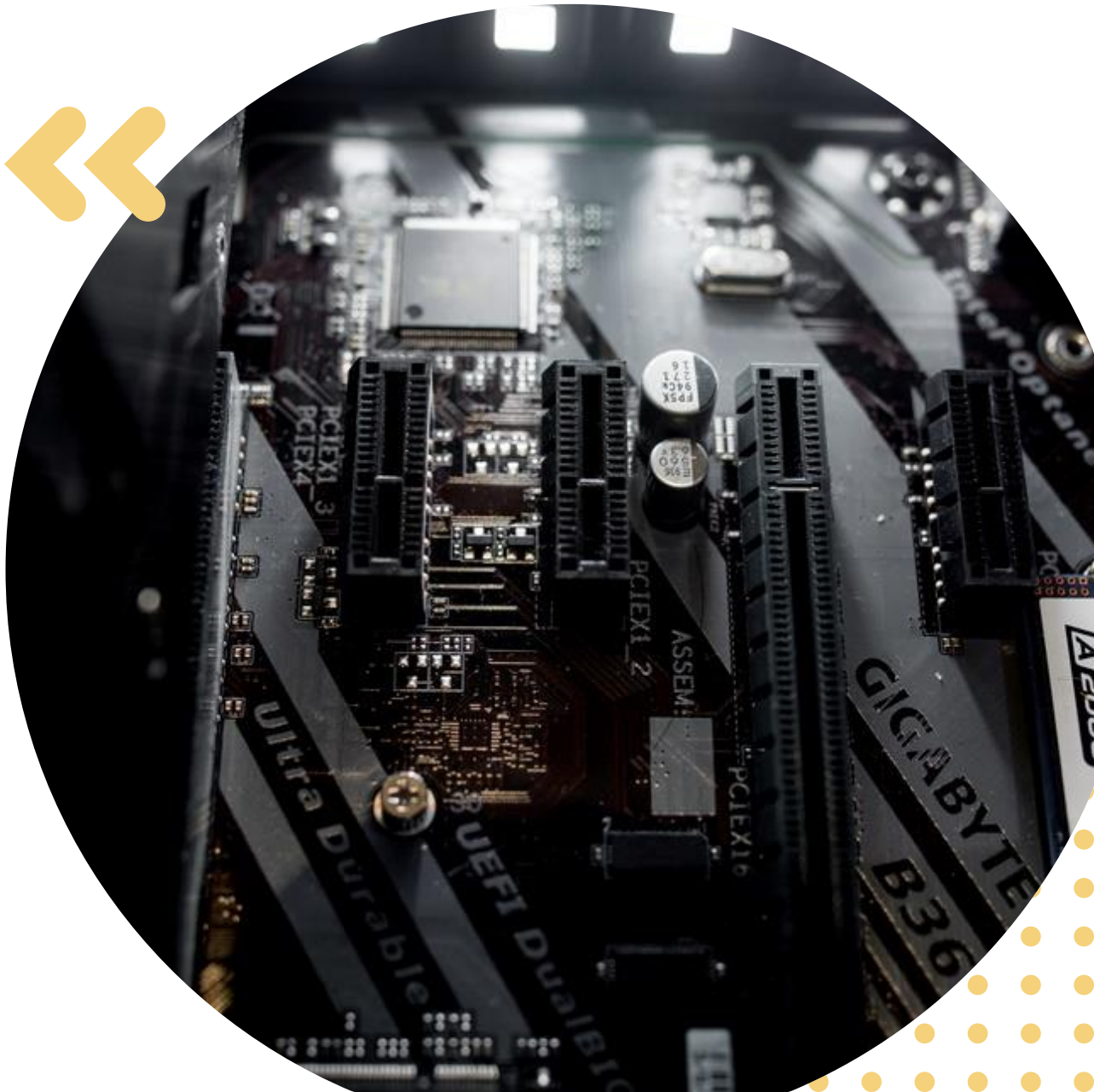
- Test environment mimics real environment
- Various methods:
  - Black box testing
  - White box testing

Black box testing

- Tests the system at a high level
- Mimics what the user will experience
- Behaviour of the system at an external level is scrutinised
- Front-end errors picked up and rectified

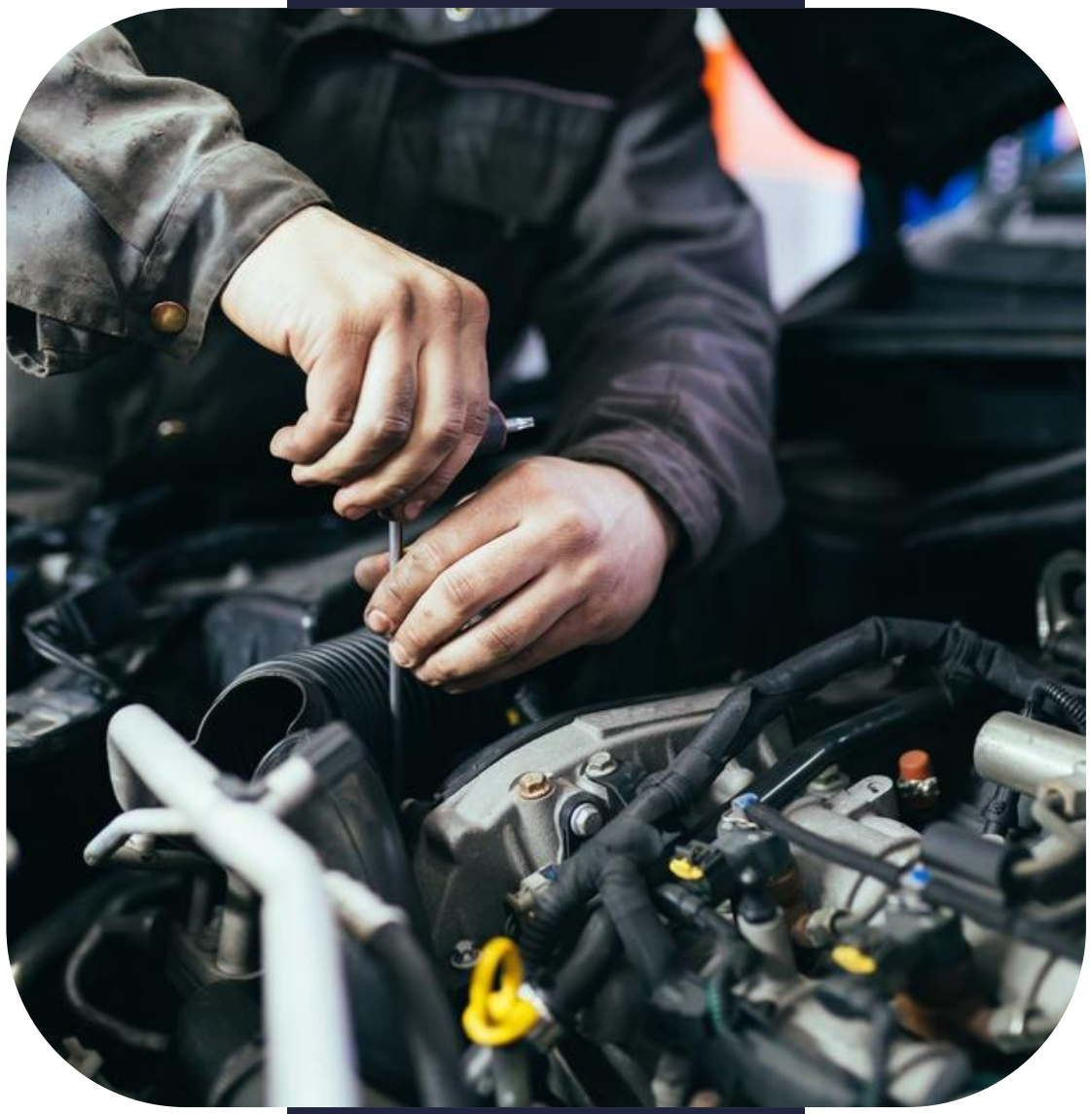
White box testing

- Looks at internals of program
- Logic, functions, and relations tested and problems rectified
- Main quality checks done at this stage



Deployment

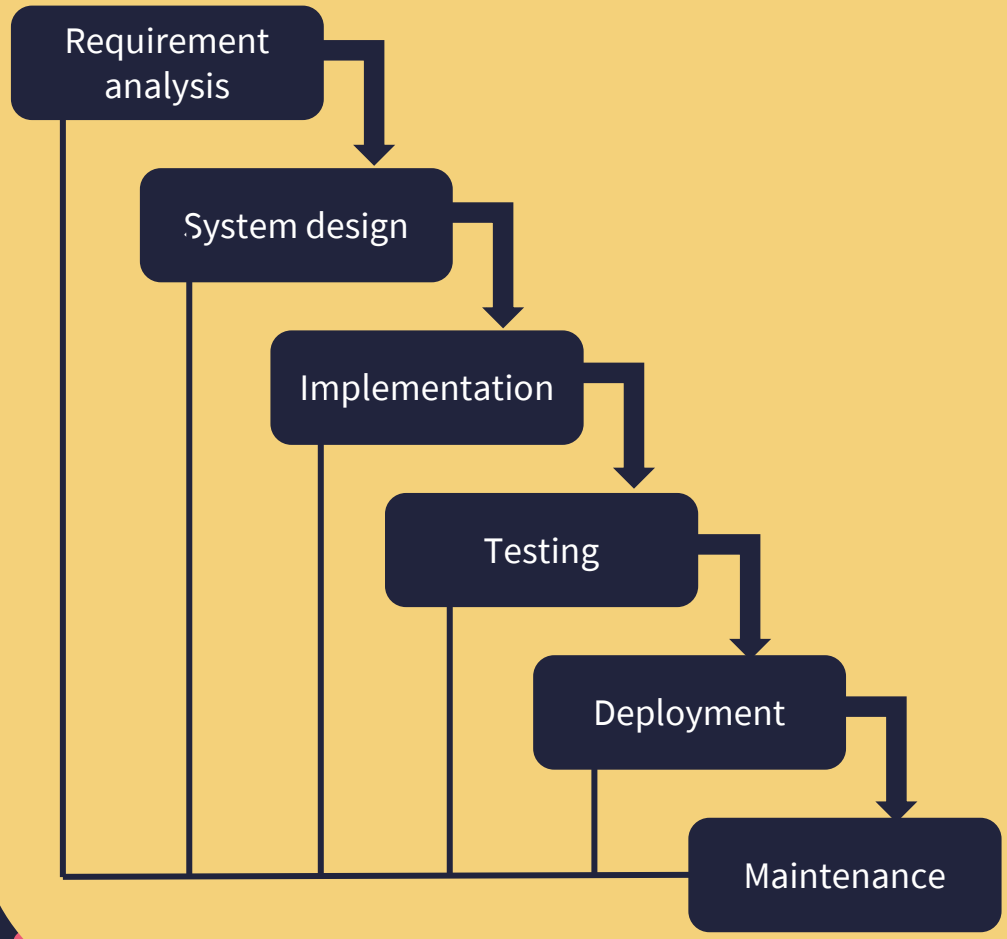
- Pilot test to see if system is running as intended
- System is then commissioned if all parties satisfied



Maintenance

- Preventive maintenance
- Perfective maintenance
- Adaptive maintenance

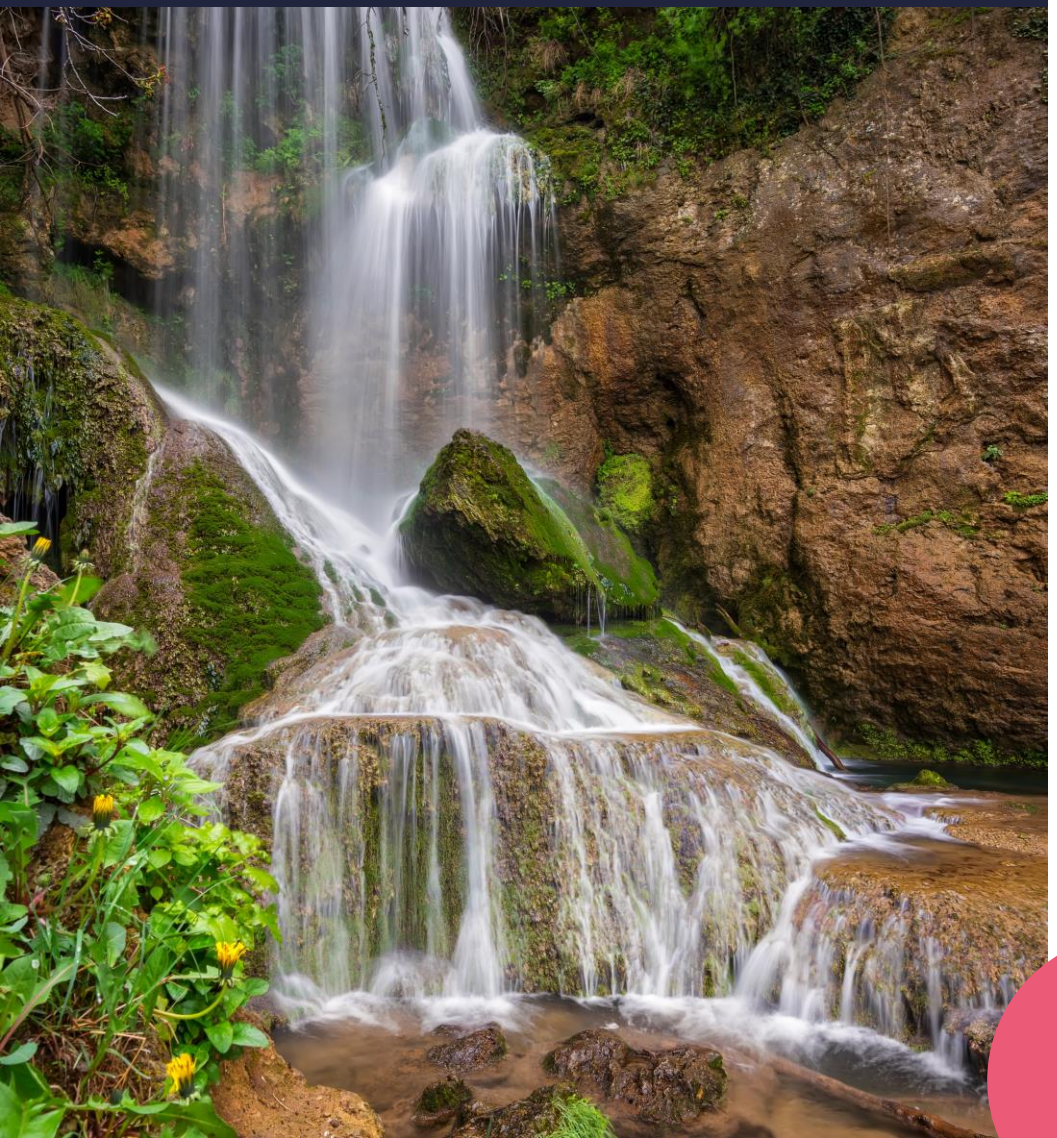
SDLC models



The waterfall model

This model was built on the premise of clearly defining each step of the software development life cycle with clear objectives and well-defined milestones.

The waterfall model structure



Each stage is a separate process.

Each stage must be completed for the next to start.

Each stage has its own deliverables; a set of tangible results and milestones.

The waterfall model structure

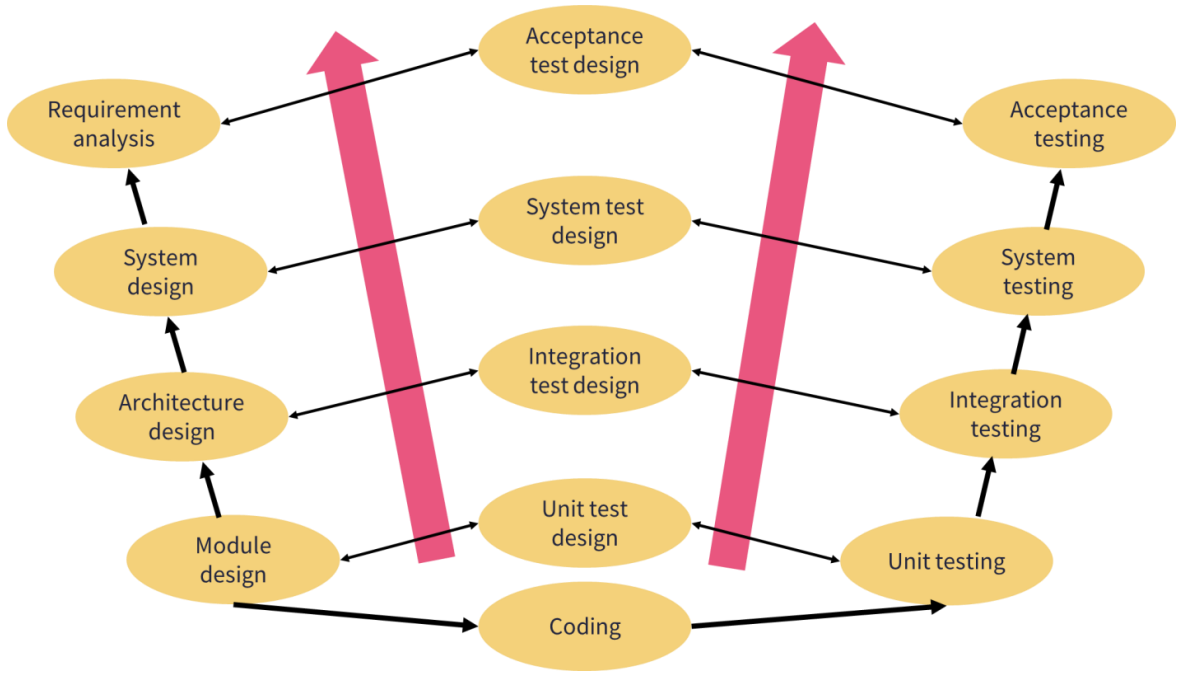
- Project is short
- Requirements are well documented and fixed
- Definition of product is stable
- Technology is stable
- Resources are abundant

The waterfall model – strengths

- A simple model that is easy to understand
- Completion of phases is logical and clearly defined
- Task are easy to arrange and distribute
- Produces a well-documented process

The waterfall model - shortcomings

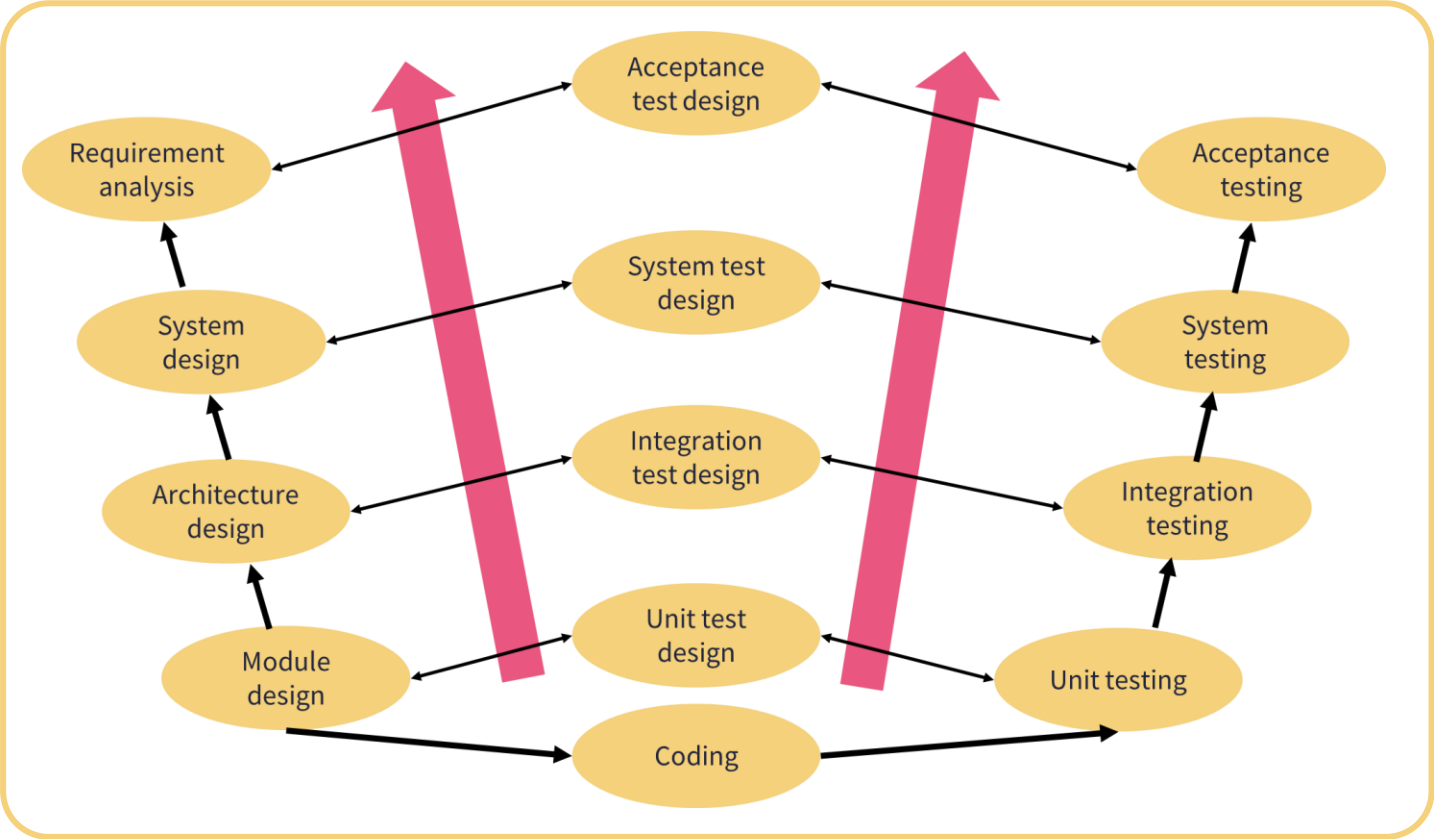
- Deliverables come quite late into the production life cycle
- Problems often undetected until it's too late
- Not suitable for complex and object-oriented projects
- Doesn’t work well for long, ongoing projects and changing requirements



The V model

- Extension of the waterfall model
- Corresponding testing phase is associated with each stage
- System development at the centre
- Same applications, advantages, and disadvantages as the waterfall model

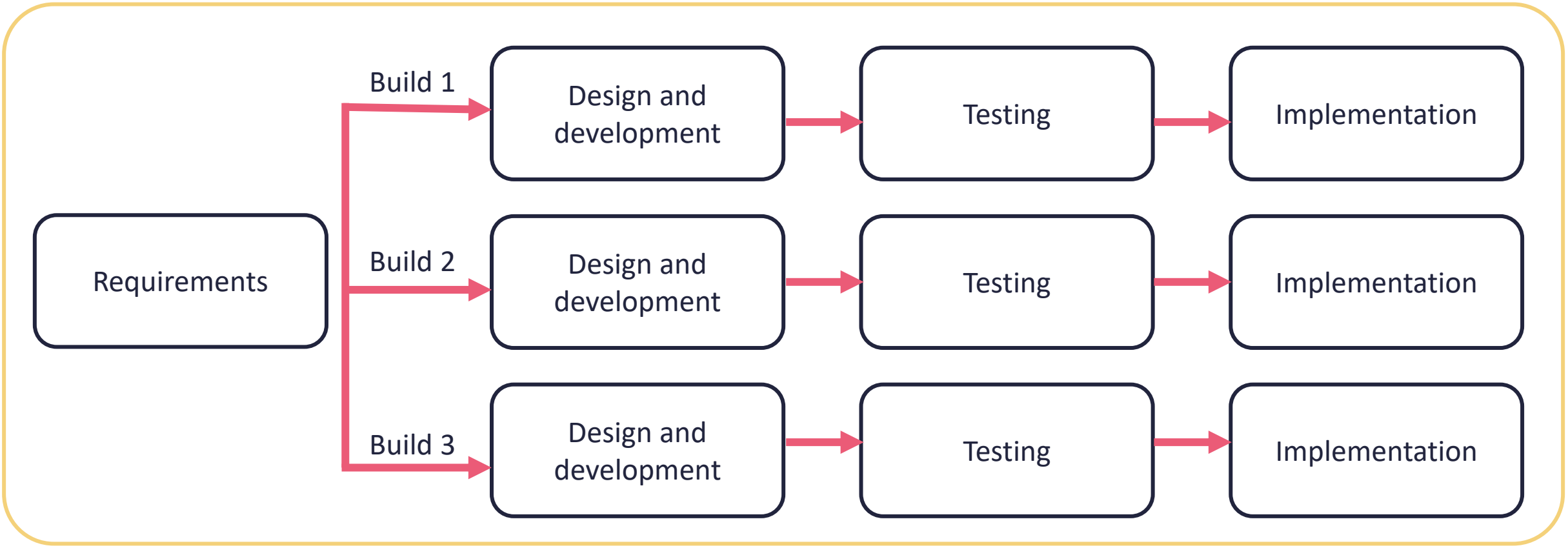
The V model



Testing carried out in 4 stages:

- **Unit testing:** code tested at an early stage
- **Integration testing**: communication of program components
- **System testing**: functionality of the entire software package
- **Acceptance testing:** compatibility on the host system

The iterative model



- Attempts to solve the problems in linear models
- Implements small subset of the requirements then iteratively builds the rest
- Repeats until all requirements are met

The iterative model - strengths

- Quick development of functionality
- Supports parallel development
- Changes to scope are less costly
- A functional product at every stage

The iterative model - shortcomings

- Requirements not as extensively gathered
- Management needs to be executed carefully
- Not suitable for small projects
- Requires skill for risk analysis

The spiral model

- Falls under the iterative model
- Mimics the natural development process
- Combination of linear and iterative models
- Emphasis on customer feedback

(Tech Receptives, 2021)

The spiral model - strengths

- Good if tight budget and risk evaluation is key
- Requires long-term commitment
- Comfortably accommodates changes
- Users can provide feedback from early on
- Development divided into chunks

The spiral model - shortcomings

- Time management can become complex
- Not suitable for small and low risk projects
- Produces a lot of documentation

More models and strategies

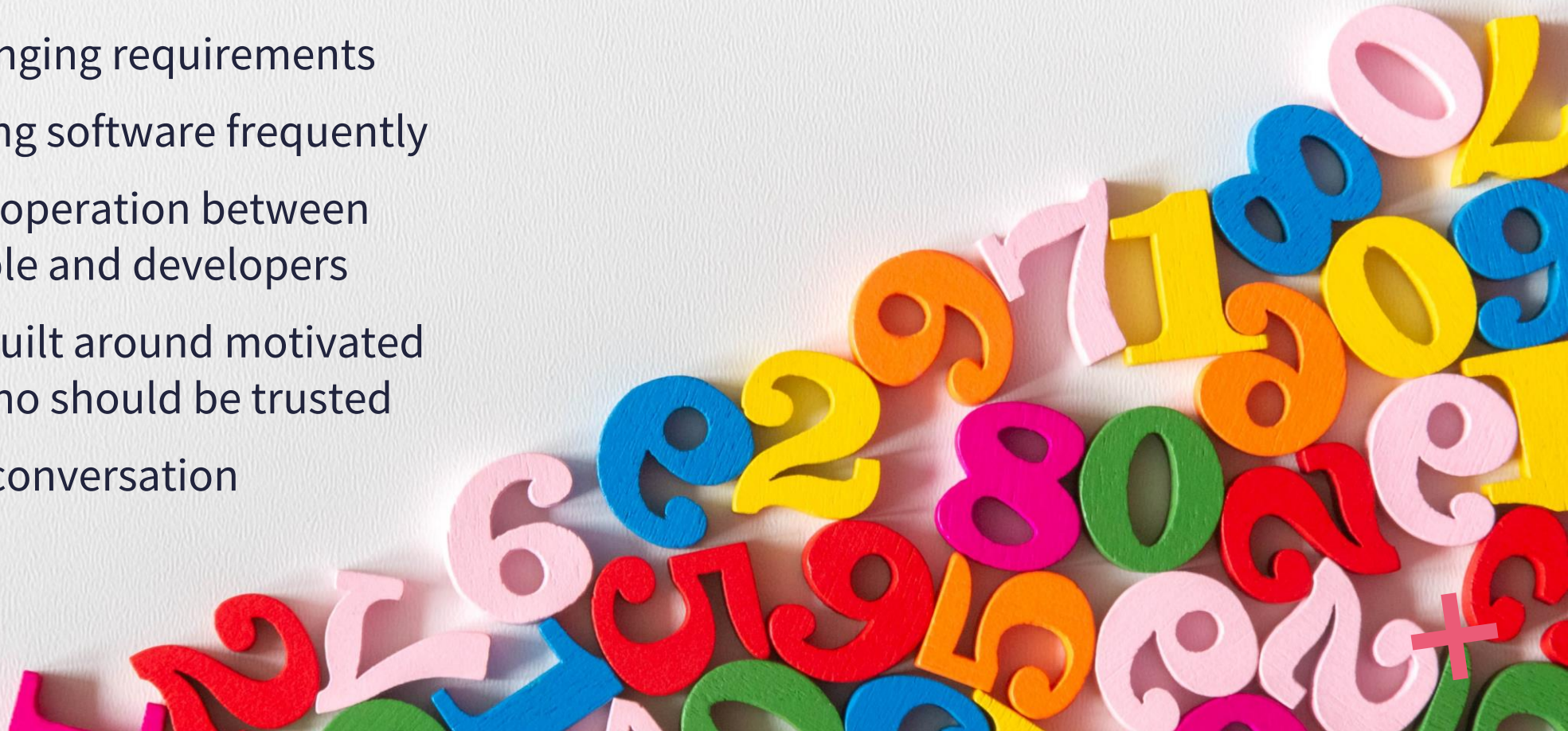
Agile development is...

A collection of cross-functioning systems built upon the iterative development process

Examples: Kanban, extreme programming, feature-driven development

The 12 principles of agile development

1. Customer satisfaction
2. Welcome changing requirements
3. Deliver working software frequently
4. Close daily cooperation between businesspeople and developers
5. Projects are built around motivated individuals who should be trusted
6. Face-to-face conversation



The 12 principles of agile development

7. Working software is the primary measure of progress
8. Sustainable development
9. Continuous attention to technical excellence and good design
10. Simplicity
11. Self-organising teams
12. Reflection

Agile model vs Old model

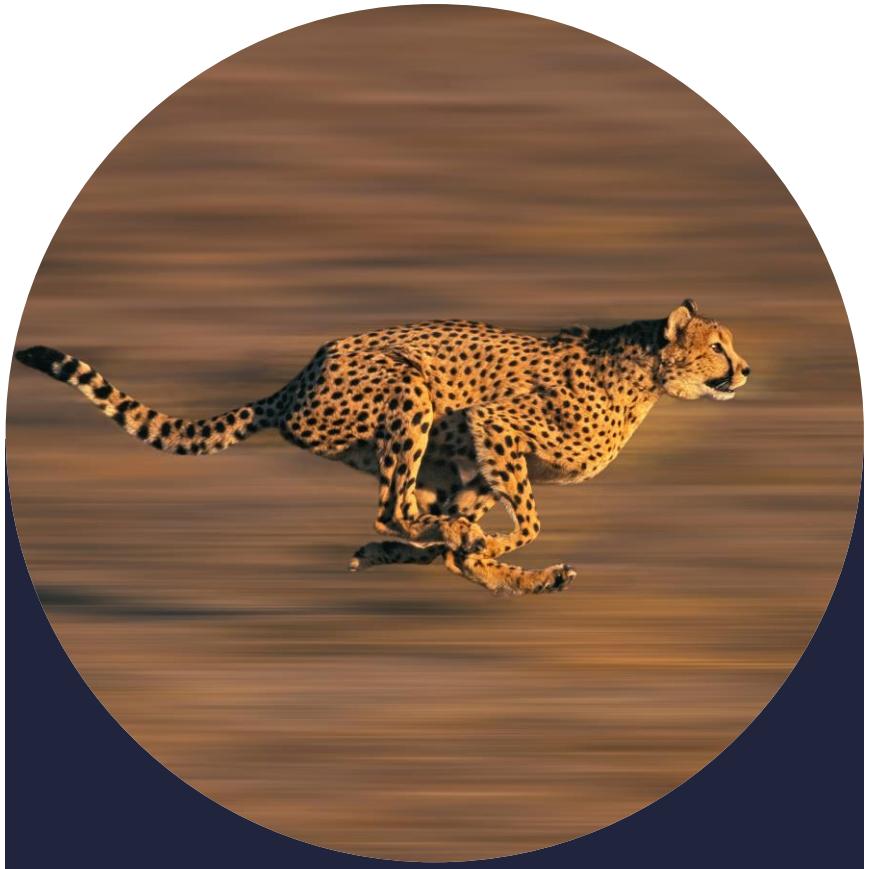
Agile model	Waterfall model
An incremental and iterative approach to software design	Development of the software flows sequentially from start point to end point
Process is broken into individual models that designers work on	Design process is not broken into individual models
Considered unstructured compared to the waterfall model	Rigid structure - customer can only see the product at the end of the project
Small projects can be implemented very quickly but hard to estimate develop time for large projects	Very plan-orientated so more secure
Errors can be fixed in the middle of the project	Problems often detected when it’s too late

Agile model vs Old model

Agile model	Waterfall model
Development process is iterative, and the project is executed in short (2-4 week) iterations	Development process is phased, and the phase is much bigger than iteration; every phase ends with a detailed description of the next phase
Documentation attends less priority than software development	Documentation is top priority and can be used to train staff and upgrade the software with another team
Every iteration has its own testing phase; it allows implementing regression testing every time new functions or logic are released	Testing phase is executed after the development phase because separate parts are not fully functional
When an iteration ends, shippable features of the product are delivered to the customer and are usable right away	All features developed are delivered at once after a long implementation phase

Agile model vs Old model

Agile model	Waterfall model
Testers and developers work together	Testers work separately from developers
User acceptance is performed at the end of each sprint	User acceptance is performed at the end of the project
Requires close communication with developers to analyse requirements and planning	Developer does not get involve in requirement and planning process – results in time delays between testing and coding



Agile processes

- Used to speed up software development
- Includes methods like scrum, crystal, feature driven development, extreme programming
- Built upon the iterative process
- Far shorter turnaround time, while producing high-quality software

Sins of software project management

- Obscurity
- Insentience
- Tardiness
- Lethargy
- Inevitability
- Obsequiousness
- Evolution

Obscurity

- When the project plan is not clear, there will be no goals in sight.
- Goals should always be measurable, specific, actionable, realistic and time based.

Lethargy

The project should be planned with clear timelines as a lack of urgency is detrimental.

Insentience

- Every project needs vison.
- Clear communication channels are vital.

Tardiness

Missing deadlines and sloppiness on milestones results in poor quality.

Obsequiousness

- Project manager must exercise their authority.
- When they don’t, problems occur such as scope creep, budget blowouts and missing milestones.

Inevitability

- When a project is executed without paying attention to risk, it is very likely to be lukewarm.
- Positive risks foster innovation, negative risks can spell doom.

Evolution

- Without due care, the scope can creep beyond the team, resources and finances.
- Make sure there is a balance between project quality and available resources.