technoteach

technocamps

Llywodraeth Cymru
Welsh Government

Prifysgol Abertawe
Swansea University

Cardiff University
Prifysgol Caerdydd

Prifysgol Bangor University

Cardiff Metropolitan University | Prifysgol Metropolitan Caerdydd

University of South Wales
Prifysgol De Cymru

Cyngor Cyllido Addysg Uwch Cymru
Higher Education Funding Council for Wales
hefcw

Prifysgol Cymru Y Drindod Dewi Sant
University of Wales Trinity Saint David

Prifysgol Aberystwyth University

Prifysgol Glyndŵr Wrecsam | Wrexham Glyndŵr University

institute of CODING in wales
technocamps

Systems Development Life Cycle (SDLC)

Systems Development Life Cycle

SDLC is a process for the planning, creation, testing and deployment of an information system. These systems may involve the development of hardware, software or a combination of both

Your learners are required to know:

- The key factors for a broad perspective of systems development
- The systemic process and phases followed by SDLC developers
- The strengths and weaknesses of different SDLC models
- Professional, legal and ethical considerations of software development

NOTE: Systems Development Life Cycle is very similar to Software Development Life Cycle which we looked at 2 weeks ago. Systems Development Life Cycle has a much broader scope, however. Be careful not to confuse them!

Systems Development – A Broad Perspective

SDLC – A Broad Perspective

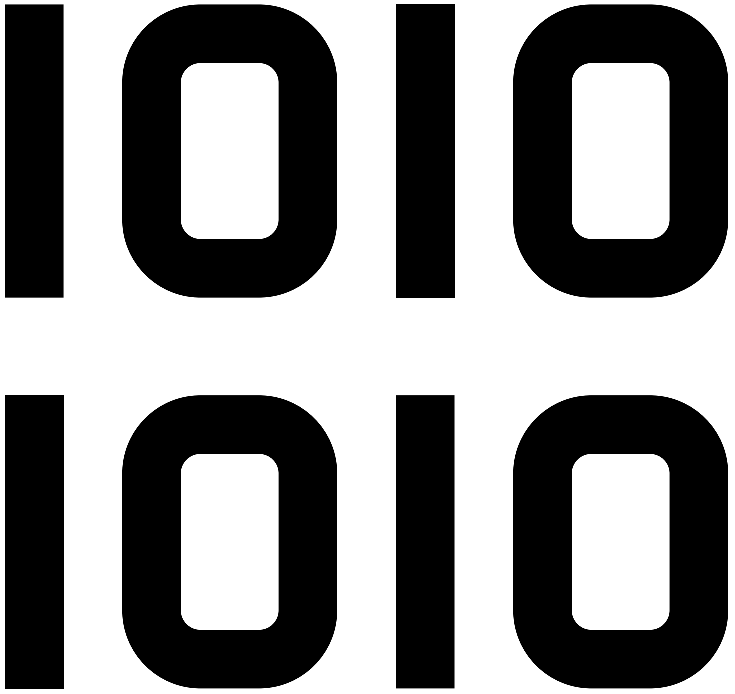
Developing a system can be a large and complex undertaking. There are various factors that may impact the processes involved. Effective systems development requires an understanding of these factors so they may be taken into account.

Key factors of systems development:

- Software
- Hardware
- People
- Processes
- Data

Software Considerations

- Requirements of the software
- User design
- Application requirements
- Database design
- Compatibility with existing systems
- Lifecycle management
- Integration of system components (e.g. databases, applications, communication tools)

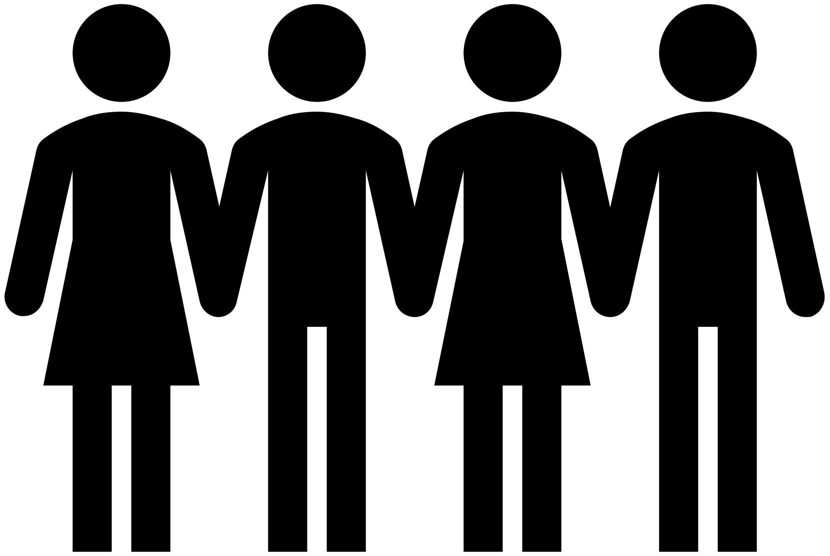


Hardware Considerations

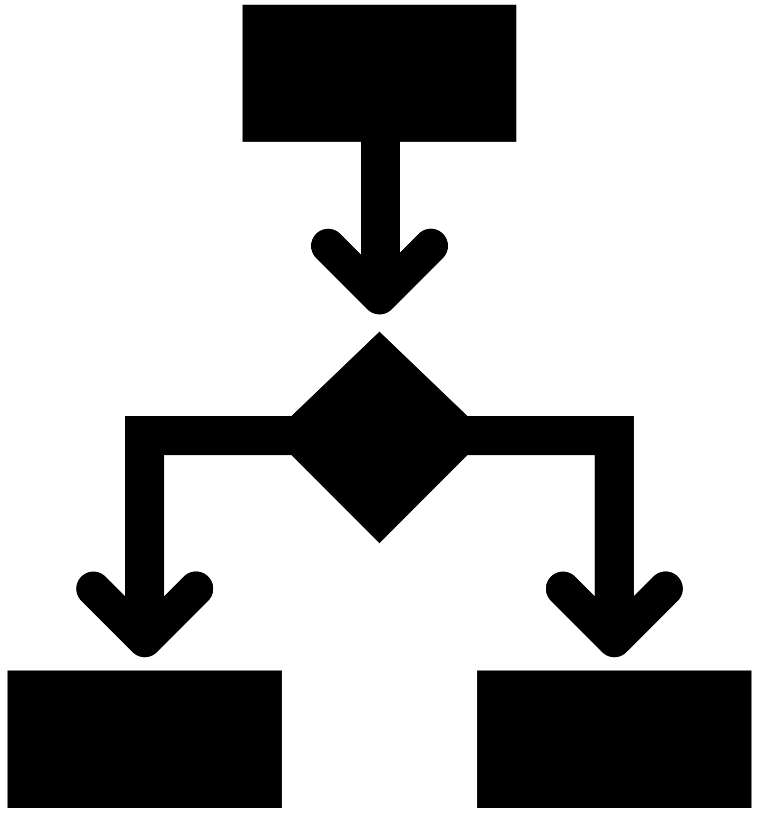
- **Infrastructure** - network, computers, peripherals
- **Optimisation** - configuration for efficiency, performance and reliability
- **Scalability** – ensuring it can meet demand
- **Maintenance** requirements

People Considerations

- **Stakeholder** interests and requirements
- **User** needs and behaviours
- **Administrators** deal with troubleshooting, updates network management and system optimisation
- Training and support by those **people that have experience and understanding** of the system.



Processes Considerations



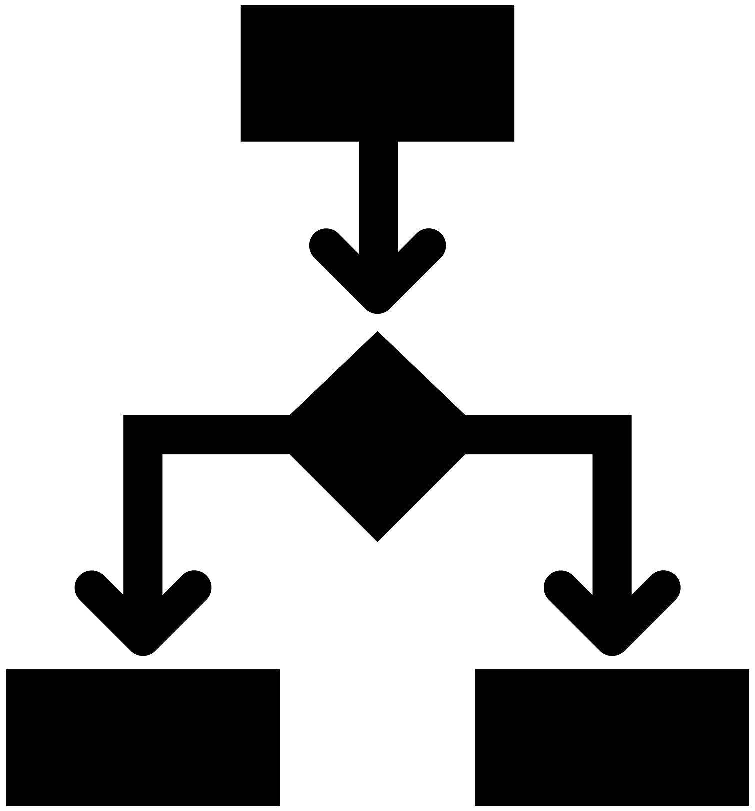
- **Workflow processes** – structured tasks that guide interactions and functioning of different system components. Ensures the system operates smoothly, reliably, and consistently

- **Automation** – increases speed and lowers costs. Automation tools take over routine activities, freeing up resources to focus on other areas

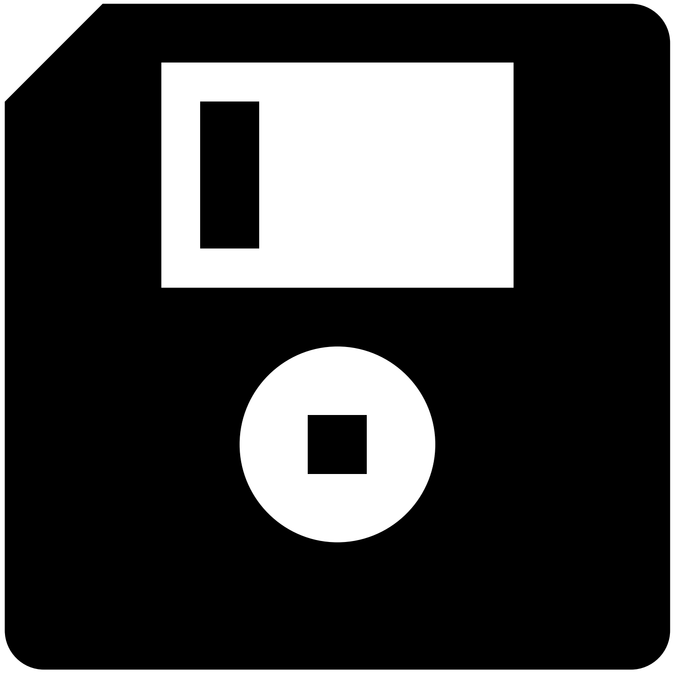
Processes Considerations

- **Compliance** with regulations and governance standards. Incorporation of checks, audits, and detailed documentation to ensure alignment with legal requirements, industry standards, and internal policies

- **Optimisation** - refining workflows and eliminating redundancies. Boosts efficiency, minimises delays, and streamlines operations



Data Considerations



- **Data Management** – collection, storage, and retrieval of data in a way that is secure, accessible, and accurate
- **Data Quality and Integrity** – validating and regularly updating data to prevent errors or inconsistencies
- **Data Integration** – enables different system components to share data and collaborate effectively

Data Considerations

- **Data Analytics** and Insights – improve decision-making, optimise operations, and enhance user experiences
- **Data Security and Privacy** – protecting data from breaches, unauthorised access, or corruption. System processes must include robust security measures, such as encryption, access controls, and compliance with privacy laws like DPA (2018) and GDPR

SDLC Process

SDLC Process

Developers in an SDLC carry out their work in 7 phases:

1. Scoping & Feasibility
2. Analysis
3. Design
4. Implementation
5. Testing
6. Deployment
7. Maintenance

These are comparable to the software development life cycle stages

Scoping & Feasibility

Objectives, nature & scope of the project are established

Alternative solutions are considered based upon interviews with stakeholders

Expected costs are weighed against intended benefits

Is the project economically, technically and operationally viable?

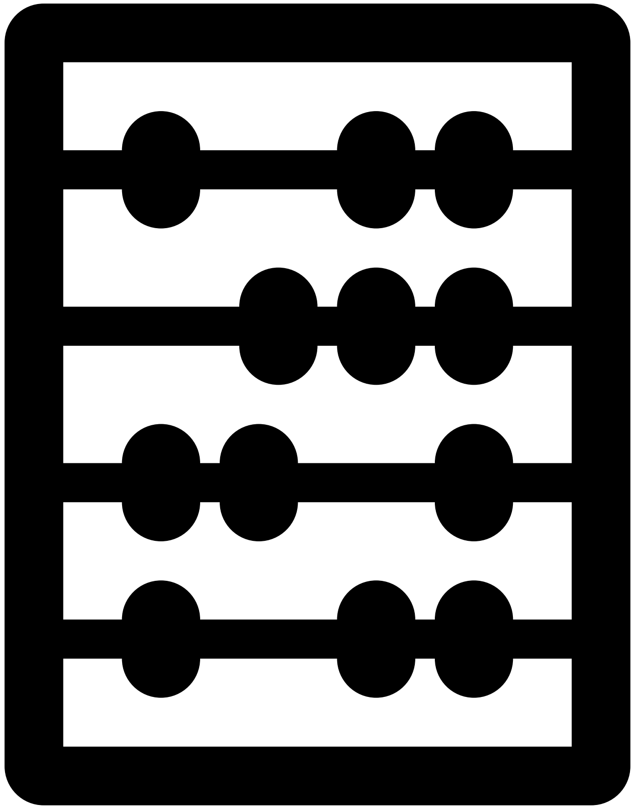
Analysis

An in-depth phase of information gathering and breaking down of project goals into system requirements

Stakeholders' needs are documented

Existing systems are scrutinised to find opportunities for improvement

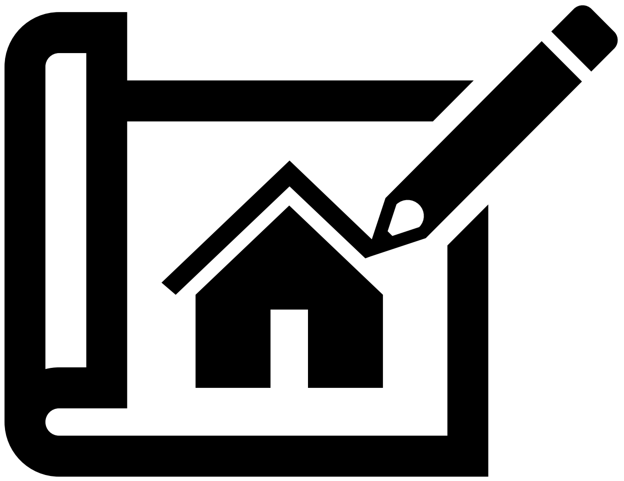
Gathered information is analysed to create a specification



Design

Desired system features are detailed:

- System architecture – how does it work?
- User interfaces - how does it look & operate?
- Data models – how is data stored and managed?



Implementation

The system is developed according to the defined requirements:

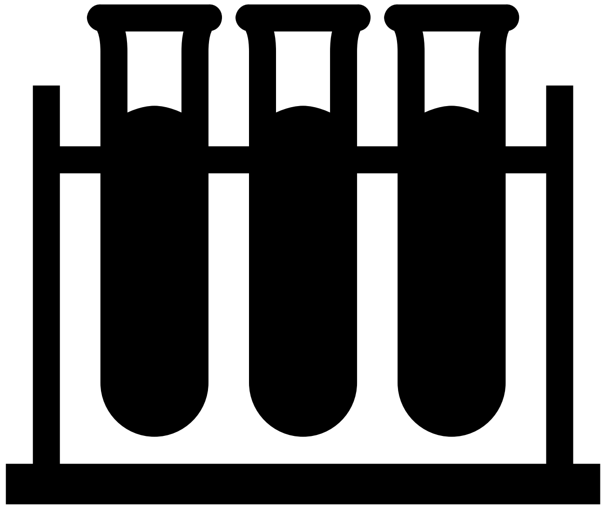
- Programming
- Hardware/software configuration
- Integration of system components

Testing

The system undergoes various tests to ensure that all components work as intended and meet the stated requirements:

- Unit testing – individual modules
- Integration testing - integrated modules
- System testing – whole system
- User Acceptance testing – does the system meet user expectations?

Any issues are fixed, then the system undergoes another round of testing



Deployment

The system is placed into production/installed

Logistics of hardware manufacturing and distribution

Transfer from previous system

Also about ensuring users understand how to interact with or operate the system:

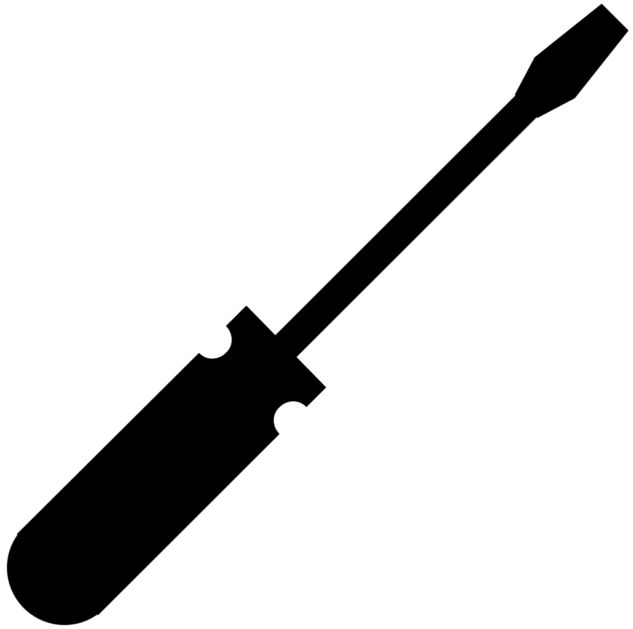
- Training
- Manual creation

Maintenance

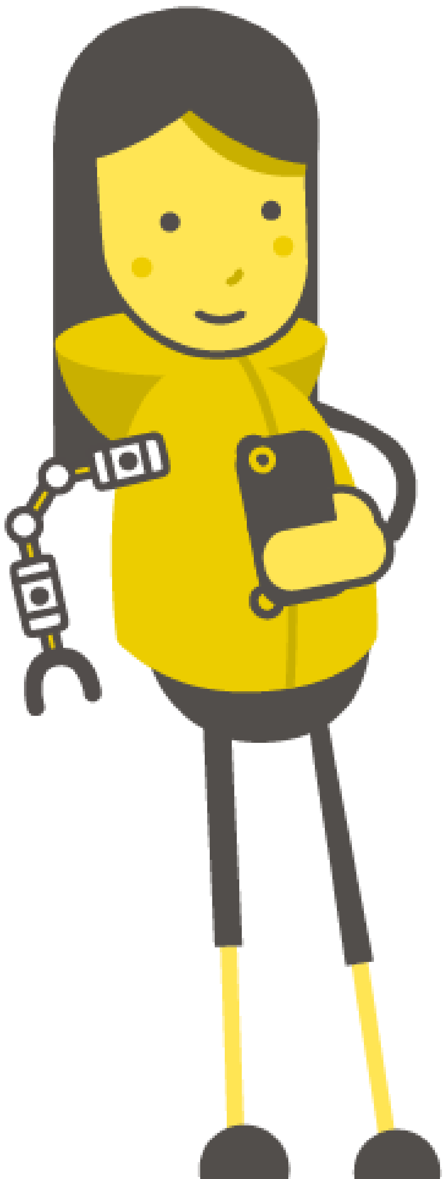
The system is continuously monitored to assess fitness

Fixes are applied as needed to maintain system quality and efficacy

Changes may be implemented to enhance the system



SDLC Models



SDLC Models

Not every company's approach to their SDLC is the same

There are various models that move through the SDLC phases in different ways, each with their own strengths and weaknesses.

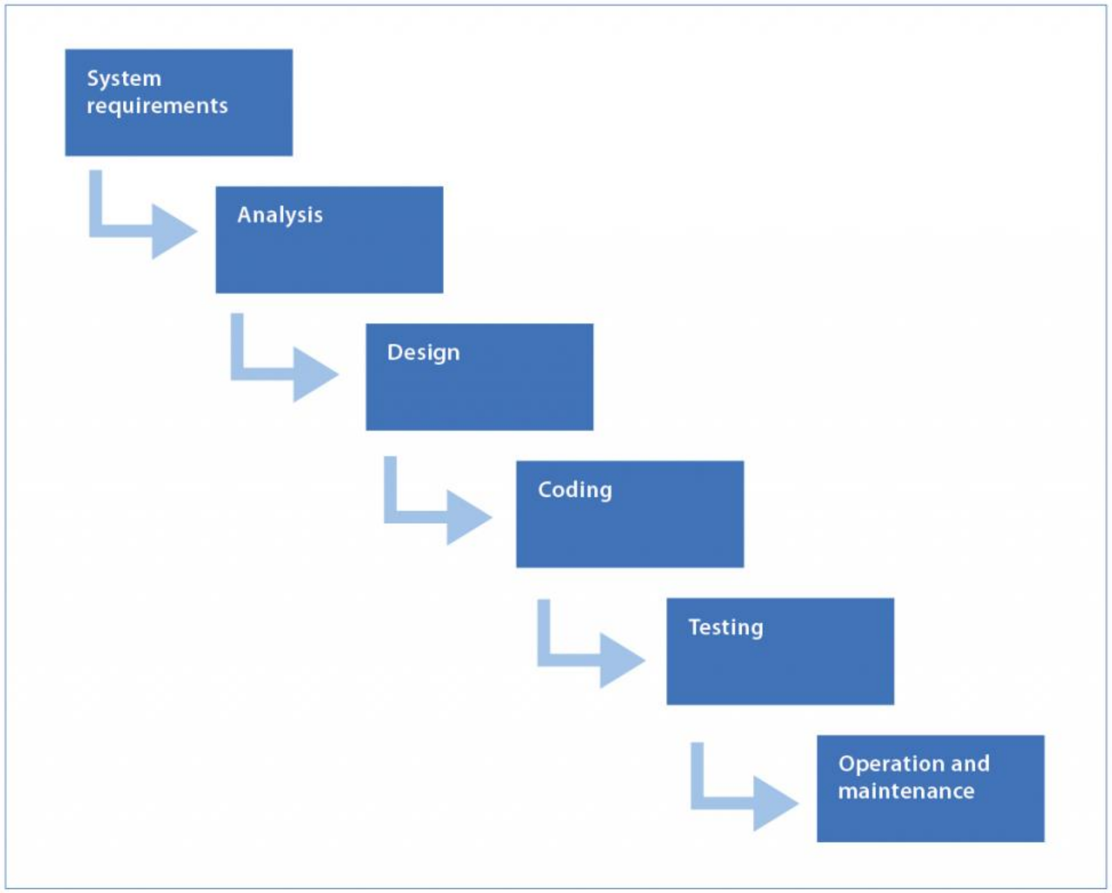
Your learners should understand the strengths, weaknesses and use of the following SDLC models:

- Waterfall
- Agile
- Iterative
- Spiral

Waterfall

Waterfall uses a linear sequence of steps – when one phase is finished, you move on to the next and do not go back

High importance is given to comprehensive planning and documentation



Waterfall

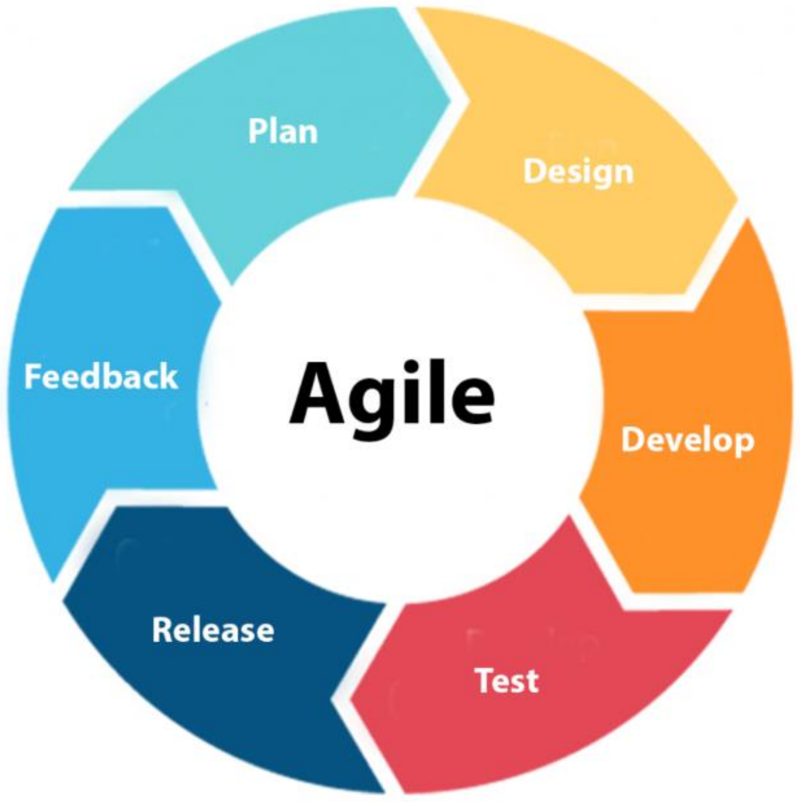
Strengths

- Clear navigation and structure
- Intuitive & collaborative
- Greater understanding of requirements from the beginning of the project
- High quality documentation enables staff to easily step into the project
- Specified timeframe & costings
- Easy tracking of progress via milestones can flag issues early

Weaknesses

- Inflexible – changes can not be made beyond the planning phases
- Limited collaboration with clients
- Only effective for smaller projects
- Working models and testing do not take place until the latter stages
- High risk if initial requirements are misunderstood

Agile



Breaks down projects into phases that end with a deliverable

There is no need to complete one section before moving on and different phases or "sprints" can be worked on concurrently.

Staff members frequently reflect on improvements to the system and workflow

An iterative and incremental process where the system is gradually grown and improved

Agile

Strengths

- Better adapts to change and requirements
- Can detect and fix issues and errors earlier
- Constant communication with clients to ensure the system meets their needs
- Removes development bottlenecking as each step need not be completed before moving on
- Staff treated as individuals

Weaknesses

- Limited documentation – harder for new staff
- Difficult to measure progress as phases blend and goalposts shift
- No full understanding of timescale or costing
- The design and long-term goals may lack cohesion as the process is fragmented

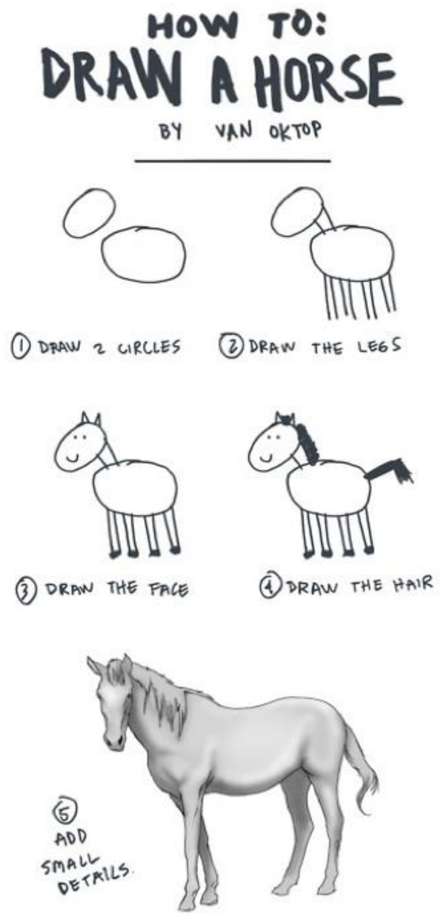
Iterative

The iterative design process is all about building and improving on the previous phase

Involves continuous improvement as the focus is to trial and improve

Less focus on research and investigation

Allows for early development and testing



Iterative

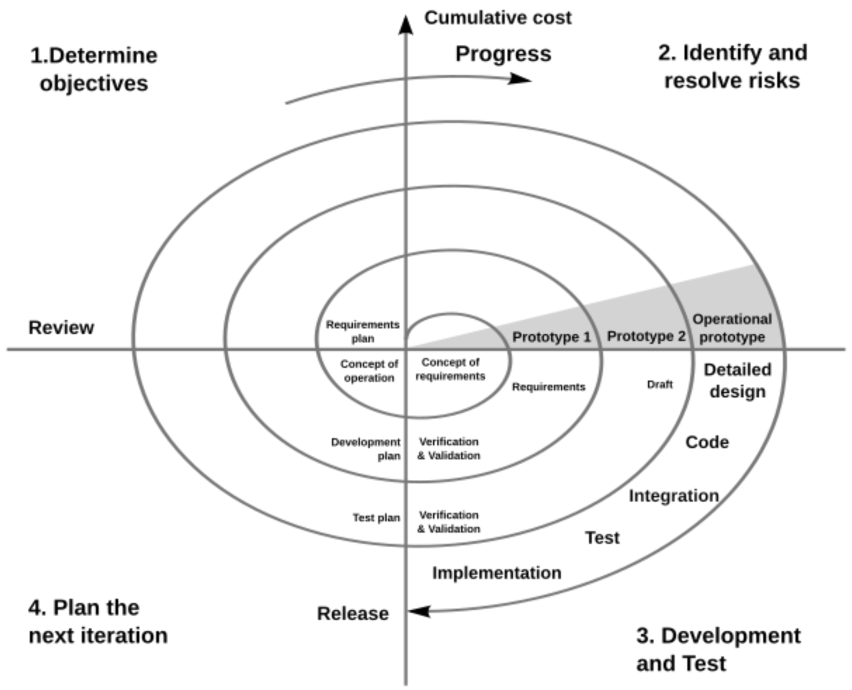
Strengths

- Frequent feedback and adjustments based on outcomes, testing and requirements
- Will have an early prototype of solution than can be reviewed, feedback and worked on
- Continuous improvement and refinement

Weaknesses

- Can be both resource and time intensive with no timescale
- May get stuck developing and refining
- Upredictable, so difficult to discuss development stages with end-users and clients

Spiral



This method of project development uses both systematic and iterative processes in its phases

The project manager will decide on the number of phases

The spiral is meant to represent the iterative processes of the system within each phase

Spiral

Strengths

- Software is produced early in the development process
- Provides opportunities for early prototypes
- Excellent for handling risk management
- Flexible – requirements may be changed during any phase
- Better for larger, more complicated systems.

Weaknesses

- Expensive in terms of resources and time – not suitable for smaller projects
- Risk of scope creep due to ongoing changes and refinements
- Difficult to estimate timescale due to the iterative element
- Requires high level expertise to the need to mange risk
- Less comprehensive system documentation produced

Professional, Legal & Ethical Considerations

Professional Considerations

Your leaners should be aware of professional considerations relating to industry standards of software development

Considerations for best practise are:

- Code quality
- Version control
- Testing
- Security practices
- User-centred design
- Collaboration & communication
- Performance optimisation
- Continuous integration & deployment (CI/CD)
- Ethical Considerations

Professional Considerations

Code Quality and Clean Coding:

- Other developers or teams may need to work on the same code. If the code is clear and well-structured, it's easier to fix bugs, add new features, and avoid mistakes.
- Developers follow guidelines like using meaningful variable names, proper indentation, and writing comments to explain complex parts of the code.

Version Control:

- Using a system to track changes made to the code over time, like using Git or GitHub helps teams work together on the same project without overwriting each other's work.
- It also makes it easy to revert to earlier versions of the software if something goes wrong.

Testing:

- Writing tests to check if the software works as expected, like unit tests or integration tests ensures that the code is reliable and helps find bugs early, preventing problems in the final product.

Professional Considerations

Security Practices:

- Developers must follow best practices, such as encrypting sensitive data, using secure passwords, and regularly updating software to fix vulnerabilities.

User-Centred Design:

- Creating software that is easy to use and meets the needs of its intended users.
- Developers work with designers and users to ensure the interface is intuitive and accessible.
- Compliance with Legal Standards:
- Ensuring that the software complies with laws and regulations, such as data protection laws.
- Developers must understand relevant legal requirements and ensure that personal data is handled safely and appropriately.

Professional Considerations

Collaboration and Communication:

- Large software projects often involve many people, including developers, designers, testers, and project managers. Good communication ensures that everyone is on the same page and that the project runs smoothly.
- Developers should attend team meetings, use collaboration tools and document their work clearly.

Performance Optimisation:

- Ensuring that the software runs efficiently, using the least number of resources as possible, such as memory or processing power.
- Developers should optimise code to improve speed and reduce the amount of data processed.

Professional Considerations

Continuous Integration and Deployment (CI/CD):

- A practice where developers integrate code into a shared repository frequently and automatically deploy the software after successful testing.
- It ensures that the software is always in a working state and allows for quicker releases of new features or bug fixes.
- Developers use tools to automate testing and deployment and must ensure their code passes all checks before it's integrated.

Ethical Considerations:

- Developers are responsible for how their software is used and must consider the consequences of their work. They should ensure that the software is used for good purposes and doesn't cause harm to individuals or society.

Legal & Ethical Considerations

Your leaners should be aware of legal and ethical considerations relating to intellectual property in software development

Legal and ethical considerations:

- Privacy
- Security
- Fairness
- Accessibility
- Transparency
- Intellectual Property
- Environmental Impact
- Social Impact
- Accountability

Legal & Ethical Considerations

Privacy

- Respect user privacy, minimise data collection, and ensure that any data collected is securely stored and protected from unauthorised access.

Security

- Develop software with robust security measures, including encryption and regular security audits, to protect against data breaches and cyber-attacks.

Fairness

- Avoid bias and discrimination in algorithms and implement measures to ensure that all users are treated fairly, regardless of their background.

Legal & Ethical Considerations

Accessibility

- Make software accessible to users with disabilities by following accessibility standards and guidelines and providing alternative input and output options.

Transparency

- Be transparent about software functionality, limitations, and risks, providing clear and comprehensive information to users about how the software operates and what data it collects.

Intellectual Property

- Respect copyrights and licenses, attribute sources correctly, and ensure that all third-party content used in the software is properly licensed.

Legal & Ethical Considerations

Environmental Impact

- Minimise energy consumption and electronic waste by optimising software for energy efficiency and supporting the recycling and proper disposal of electronic devices

Social Impact

- Consider broader societal implications of the software, promoting positive outcomes such as social inclusion, education, and well-being, while mitigating any negative impacts

Accountability

- Take responsibility for the ethical implications of the software, promptly addressing and rectifying any issues that arise, and ensuring that there are clear processes for users to report concerns