LECTURE 2

Selection of software development lifecycle model

Content

1. Software development lifecycle (SDLC) revision
2. Some typical SDLC models
3. Selecting appropriate SDLC model for software project

Software Development Lifecycle

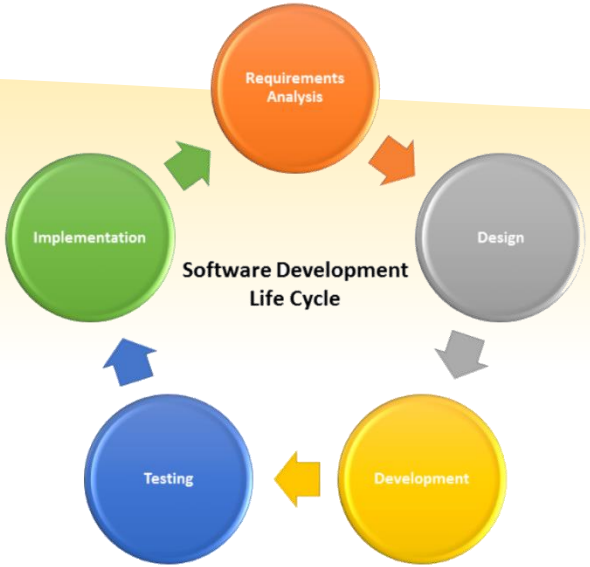
- When software systems were first developed in the mid twentieth century they were done so in a rather ad-hoc manner without any reference to a defined development process.
- It was soon realized that this undisciplined approach led to systems that were poorly constructed and difficult to manage and control.
- Some form of development process was required that could assist software developers in producing more structured code in a more manageable way.

Software Development Lifecycle

- The advantages of structuring software development into a defined process model are that:
  - It divides a large problem down into easy-to-understand tasks at each stage.
  - It sharpens focus.
  - It supports planning and control – improves time estimates.
  - It provides progress visibility.
  - It provides structure.
  - It leads to better coding and documentation.

Software Development Lifecycle

- The Software Development Life-Cycle (SDLC) represents a generic model for software development and consists of a number of stages including requirements analysis, design, development, testing and implementation.

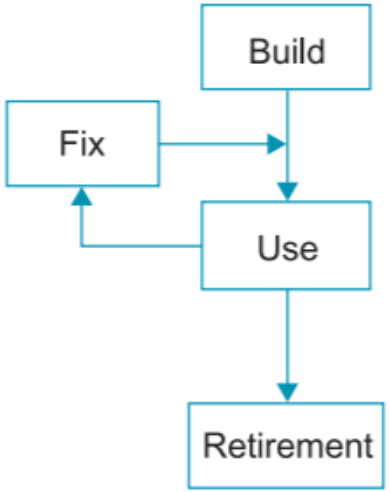


The ‘Build-and-Fix’ Model

- In the pioneering days of software development there was no recognized process for developing software.
- The build-and-fix or code-and-fix ‘model’ represents this early approach to the development of software.
- There is no formal requirements capture in the process and no formal design. Programmers would first write some code, run the code and then correct any bugs in the software.
- In this case there is no formal breakdown of the process into stages.

The ‘Build-and-Fix’ Model

- The model merely iterates until the software becomes unworkable and is eventually retired and/or replaced.



The build-and-fix ‘model’

The ‘Build-and-Fix’ Model

- Disadvantages:
  - ✓ After several fixes the software becomes difficult to maintain as it becomes poorly structured.
  - ✓ It often does not match the user’s requirements – it is rejected or requires extensive redevelopment.
  - ✓ It can be costly to maintain because of its poor structure and lack of definable output that can be tested.
- Although the build-and-fix approach is still used today by many programmers working on small and personal systems, you should not be using it for your project.
- You should, instead, look to one of the more defined and recognized processes.

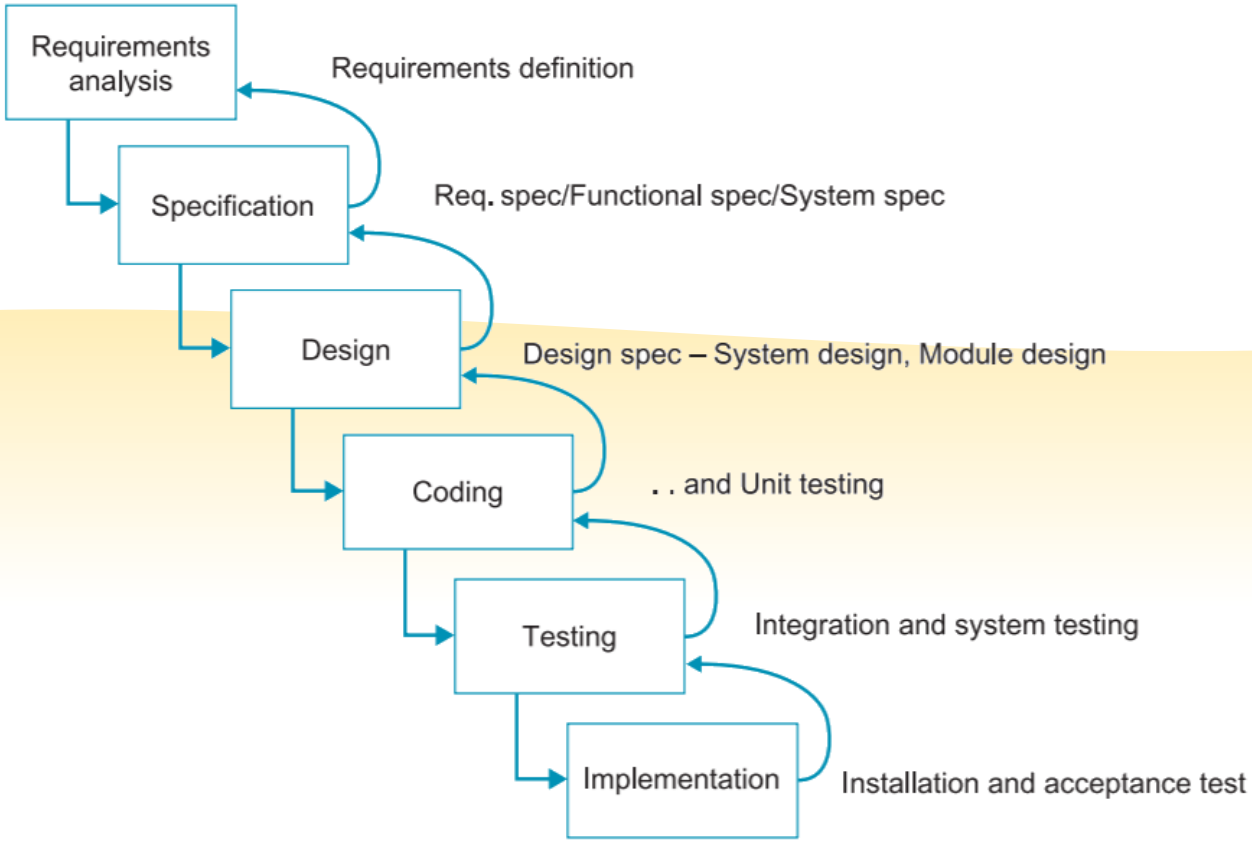
The Waterfall Model

- Owing to the problems encountered with the unstructured approach of the build-and-fix model, several more detailed models were devised.
- The earliest of these models was the stage-wise model from which the waterfall model developed.
- The stage-wise model was developed in 1956 by Benington in an attempt to provide an engineering process to the development of software.
- It represents a unidirectional, sequential process – once a stage has been completed, the results of that stage become a fixed baseline from which the following stages develop – there is no revision.

The Waterfall Model

- However, that the sequential nature of the stage-wise model was causing problems and some form of reworking was required to allow a user's needs to be addressed more effectively. For this reason, the Waterfall model was developed.
- The Waterfall model differed from the stage-wise model in that it allowed some limited iteration between stages – shown as 'splashing back'

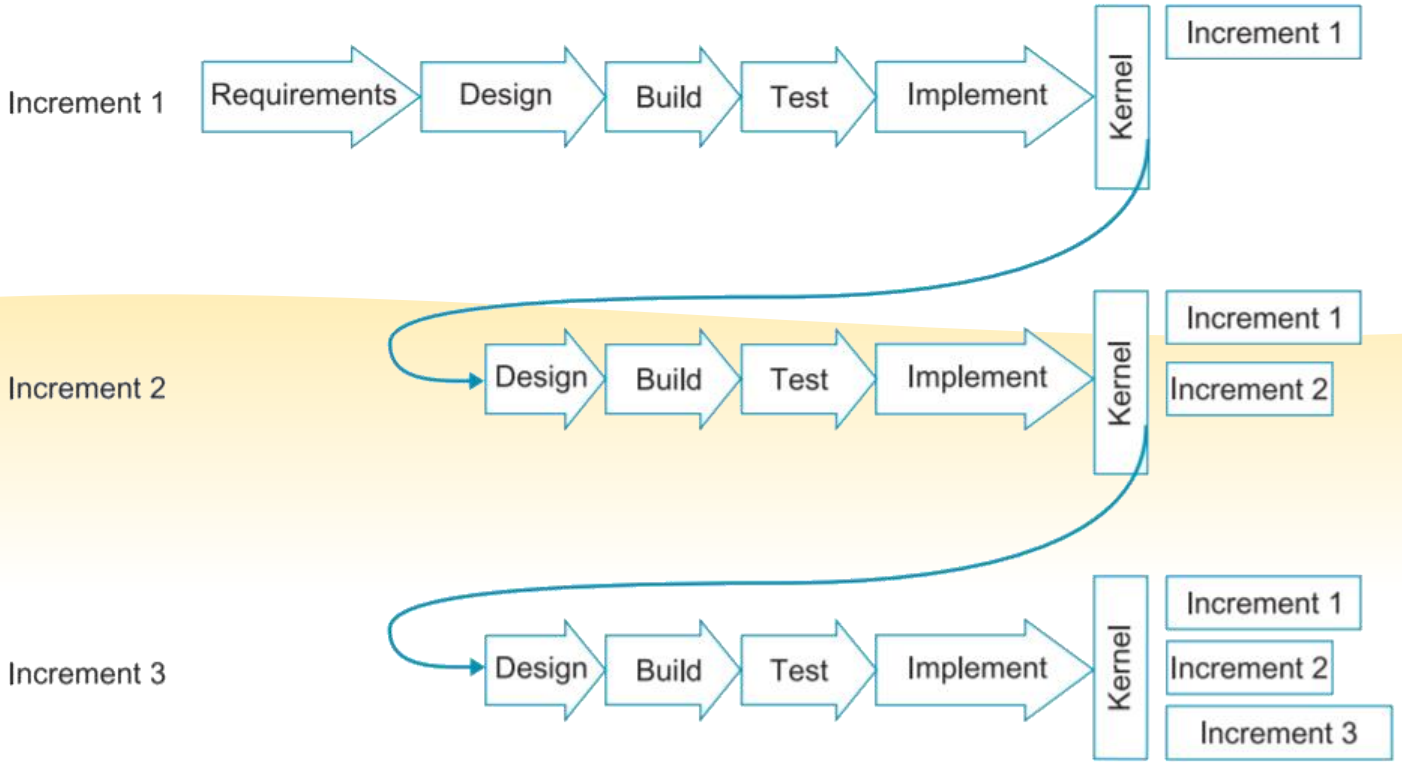
The Waterfall Model



The Incremental Model

- Rather than delivering your software to your client towards the end of your project as the conventional models do, it might be better to deliver the system to them as a series of intermediate working subsystems over a period of time.
- Thus, you add more functionality to the software at each release of the system. This means that you need to get an overall software structure (kernel) in place as part of the first release of your system.
- The other parts of the system are then brought online and released to the user as the system is developed. Thus, each release to the user provides added functionality to the existing system.

The Incremental Model



The Incremental Model

- Advantages:
  - The user gets something early so that they can get an idea of the system's capabilities and an idea of what you are able to produce in the longer term for them.
  - Delivering something early gives you a sense of achievement and the client/user a clear understanding that progress is being made.
  - It can help you plan and manage your project more effectively.
  - The user does not need to learn how to use the entire system in one go.

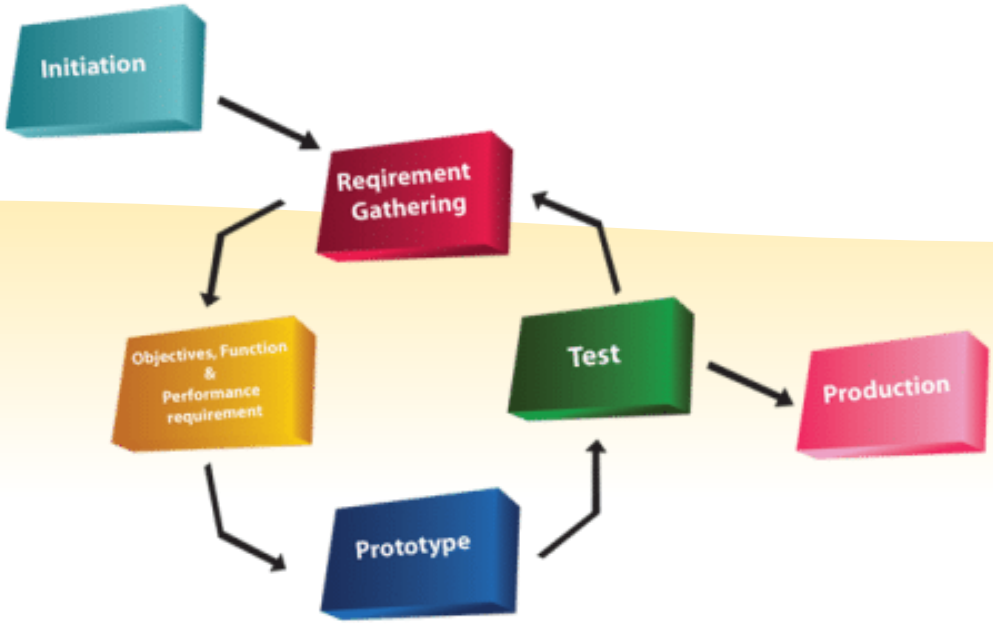
The Incremental Model

- Disadvantages:
  - It might be difficult to break your program down into a series of subsystems that are worth delivering to the client/user as separate units.
  - Although it is advantageous to meet your client/user regularly, additional contact with them can encourage them to identify too many improvements. They may identify more changes than you have time to implement and the changes they request may take your project in a direction that is inconsistent with the requirements of your course.

Prototyping Model

- In many projects it is often difficult to pin down exactly what is required from a software system at the start of the project and/or we may not have a clear understanding of the technical issues surrounding that system.
- In these cases it is useful to produce a prototype in order to:
  - Explore the requirements of the system with the user – requirements capture.
  - Explore the concepts for a proposed system – conceptual prototyping.
  - Explore the technical feasibility of a system – experimental prototyping.

Prototyping Model



Prototyping Model

- There are two things you can do with a prototype once you have developed it:
  - You can throw it away and start the development of the system from scratch (***throw-away prototyping***).
  - Or you can develop (evolve) the prototype into the final system (***evolutionary prototyping***).

Prototyping Model

- Advantages:
  - Something tangible is produced quickly which keeps you and your user happy.
  - The tangible nature of the system helps the user to clarify their ideas and refine their requirements.
  - Misunderstandings, errors and omissions in the requirements can be sorted out.
  - It improves communication with the user. It is easier to look at a working model than a document.
  - The prototype can test the feasibility and usefulness of the product. Alternatives can be compared using different prototypes

Prototyping Model

- Disadvantages:
  - The user might think your prototype is really good and may want you to develop it into the final system, even though you had intended to throw it away. As a consequence you will end up building your final system on perhaps poorly structured code.
  - If you develop the throw-away prototype on a different system or in a different programming language you may not spot a technical issue that will be difficult/impossible to overcome on the target system.
  - It can take a lot of effort and commitment to develop something that you are eventually going to discard.

Agile Methods

- The term Agile methods was adopted in 2001 by a group of eminent software engineers at a meeting in Utah, USA.
- It refers to approaches to software development that reduce risk by delivering software systems in short bursts or releases.
- The other main characteristics of Agile methods that differentiate them from older: their emphasis on smaller development teams (which are invariably working together in open plan offices); and face-to-face communication with the users who are quite often based in the same working environment as the developers.

Agile Methods

Agile Methods

- Agile methods are well suited to projects that have unclear or rapidly changing requirements; the project team is fairly small but highly competent and can be trusted; and close interaction with the user must be possible.
- A number of key principles include:
  - ✓ Satisfying the customer through early and continuous delivery of valuable software
  - ✓ Welcoming changing requirements.
  - ✓ Short iterative timescale of weeks.
  - ✓ Close working relationship between developers and users.

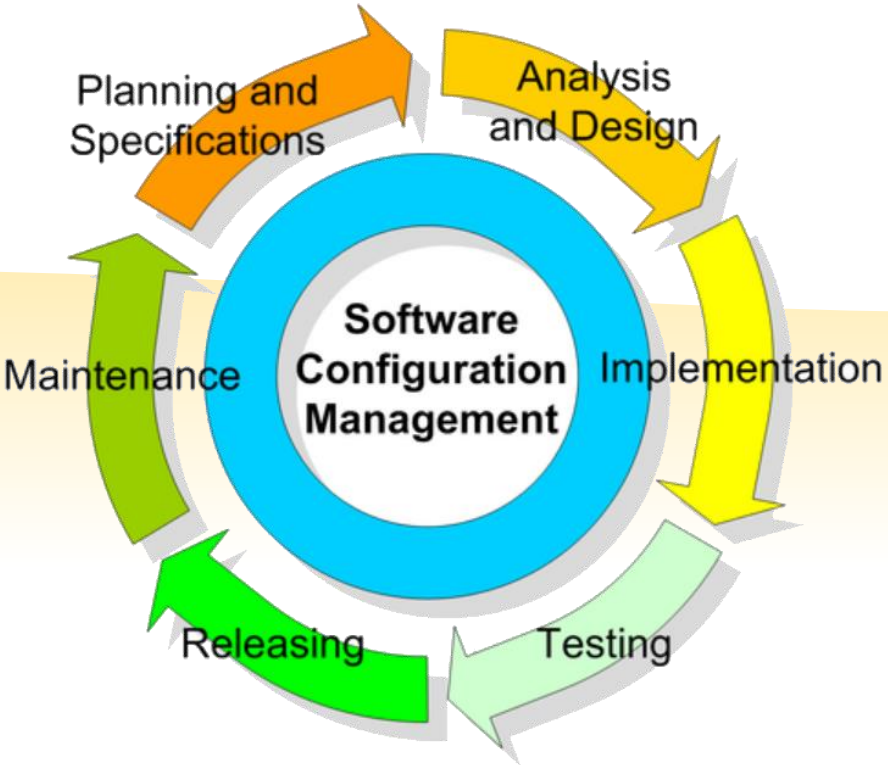
Agile Methods

- A number of key principles include:
  - ✓ Face-to-face conversations rather than detailed documentation.
  - ✓ Working software is the measure of progress.
  - ✓ Teams are self-organizing.
  - ✓ Teams reflect on how they might improve themselves regularly.
- Because of the reduced amount of documentation agile methods produce, they have come in for some criticism.
- However, for smaller projects the principles can lead to the development of software that goes a long way to meeting the users' requirements.

Configuration Management

- Configuration management is used to control the different versions of a system that are produced as the system develops.
- It is closely related to version control and revision control.
- For student projects a comprehensive configuration management system is unnecessary.
- However, as you develop and release new versions of the system you must make sure that you keep track of these versions; what changes have been made, who has made these changes, who has received them, where they are saved/backed-up to, etc.

Configuration Management



Configuration Management

- In large industrial projects configuration management is an important activity and typically involves four stages:
  1. Configuration identification: identifying the attributes that define the item you are hoping to control – what makes it as it is – and setting this as a baseline.
  2. Change control: managing and approving changes to the item and adjusting the baseline.
  3. Status accounting: recording the configuration baselines.
  4. Configuration audits: ensuring that changes to configuration do what they state they will do and the system continues to satisfy the requirements.

Configuration Management

- If configuration management is going to be an important part of your project it may be worth considering using a software tool to support this. An example of such a tool is TortoiseSVN.

Which Approaches Should We Use?

- **Waterfall Model**:
  - This is well-suited to projects that have a low risk in areas such as the user interface and performance, but a high risk in terms of schedule and control.
  - It is useful in projects where the requirements are understood clearly by all involved and are unlikely to change during the course of the project.

Which Approaches Should We Use?

- **Incremental Model**:
  - This is appropriate for systems in which the requirements are generally clear and the system can be partitioned into a series of intermediate deliverables.
  - It is preferable to the conventional approach in that it provides some user involvement and it can also be ‘terminated’ at any time yet still leave the user with a partially working system.
  - However, if the user is difficult to contact and you can only arrange one or two meetings with them, the conventional waterfall-type model may have to be used.

Which Approaches Should We Use?

- **Throw-away prototyping model**:
  - This is useful for ‘proof of concept’ situations, exploring technical uncertainties, or in situations where requirements and user’s needs are unclear or poorly specified.

Which Approaches Should We Use?

- **Prototyping model**:
  - This is useful in projects that have a high risk in terms of the user interface design and uncertainties in user requirements.
  - It is perhaps more appropriate than a throw-away prototype in situations where the programming language and target system are well understood by the student.

Which Programming Language Should We Use?

- There is no simple answer to this question since the choice of which programming language to use in your project depends, not only on the application you are developing, but on a number of other issues too.
- Deciding on which language to use is primarily based around who the client/user of the system will be and the platform on which the software is based.
- The following list of factors that should be taken into account when making your decision:

Which Programming Language Should We Use?

- What languages do I already know – how good am I at programming using those languages?
- Are there any languages I would like to learn as part of my project?
- What language(s) does my supervisor use – will I need a lot of technical support from him/her?
- What languages are supported by my department or others within my institution?
- If I develop a system using a language that is not supported by my own department, will I be penalized if things go wrong?

Which Programming Language Should We Use?

- If things do go wrong with a language that is supported by my department, and my project goes badly as a result, who will be responsible?
- If I need to get technical help or support, is it readily available (for example, either locally or via the Internet)?
- Are there any language requirements imposed on my project by the client/user?
- What languages are supported by the client/user?
- How much does the nature of the system I am developing affect the choice of language?

THANK YOU !