UNIVERSITY OF LIVERPOOL

Application Development with C++ (ELEC362)

Lecture 12: Basics of software development

mihasan@liverpool.ac.uk

Previous lecture

- Iterators were introduced an the different types of iterators were discussed.
- Stream iterators and insert iterators were discussed.
- Smart pointers were discussed.
- Algorithms and Functors were defined and discussed.

This lecture

- What is covered in this lecture?

  1. Models of software development. 2. Discussion of their components.

- Why it is covered?

  Because it provided a high-level out-take on the software development process.

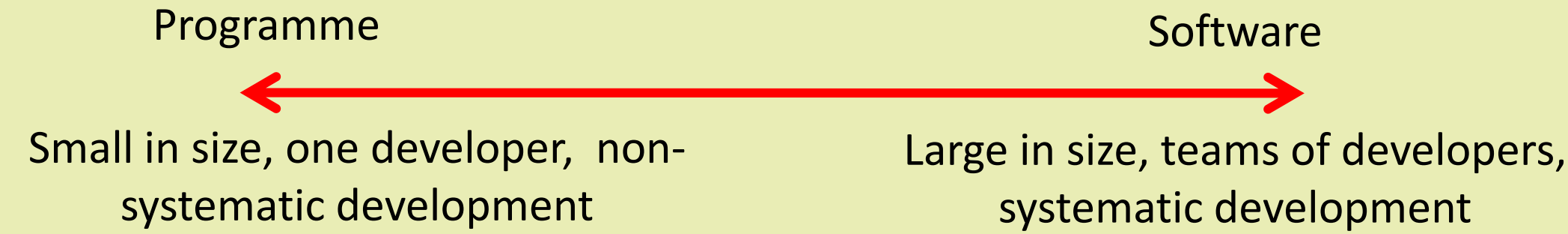
- How are topics covered in this lecture:

3 practical examples

NOTE: This is NOT a Software Engineering module in one lecture.

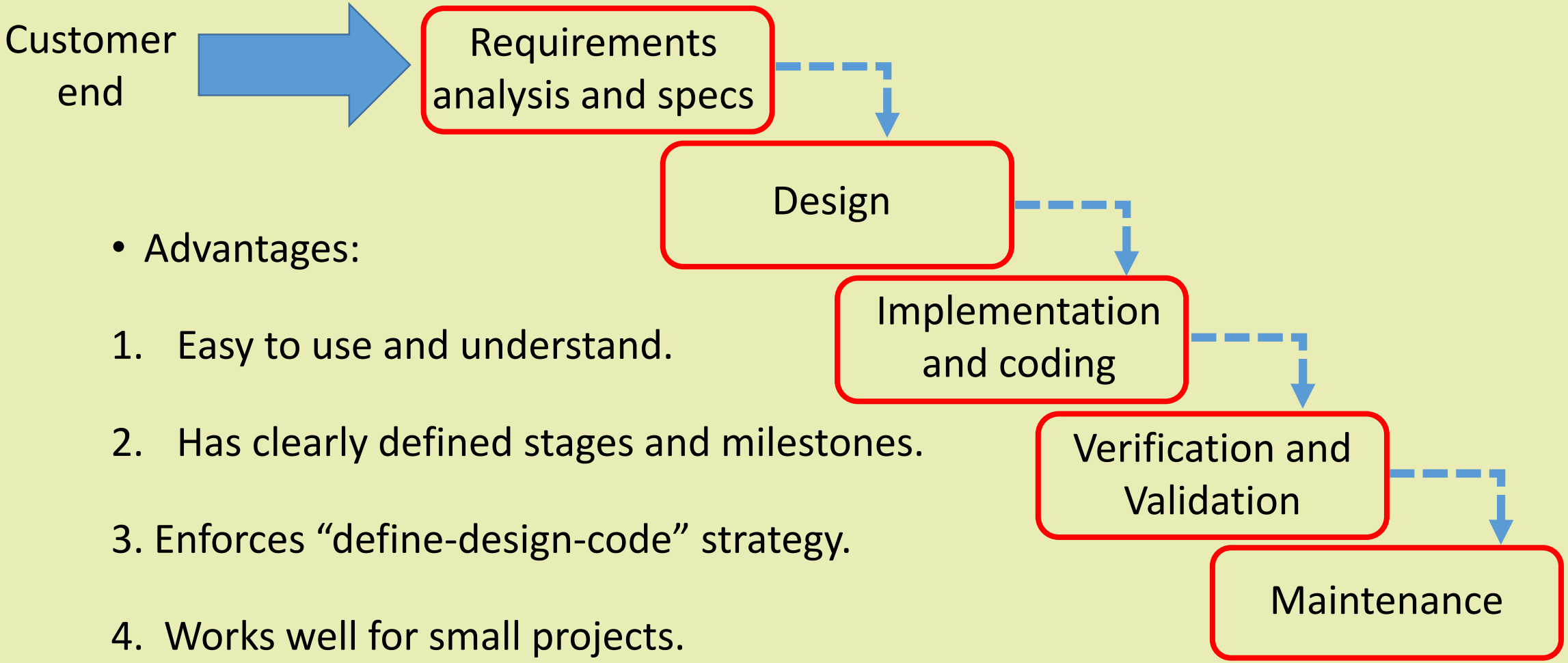
Software Engineering

- Software Engineering is a systematic approach to the design, development, operation, and maintenance of a software system.
- Software engineering concepts are important in defining the overall objective of a given software, in comparison to codes which are very specific.
- In Software engineering, the words “programme” and “software” mean two different things:



The waterfall model

- The waterfall model is the most basic model in software engineering.



- Advantages:

1. Easy to use and understand.
2. Has clearly defined stages and milestones.
3. Enforces “define-design-code” strategy.
4. Works well for small projects.

Requirements analysis and specs

- The customer gives the specifications of the software, in order for the developer to produce the Software Requirements Specification (SRS).
- A Software Requirement Specification (SRS) is description of a software system to be developed, stating functional and non-functional requirements.
- The SRS is a standardised document for software projects, with one of the most famous standard being the IEEE Std 830-1998 (https://ieeexplore.ieee.org/document/720574).
- Mention that you are aware of it in your cover letter for software development jobs!!

Requirements analysis and specs

SRS Requirements

Functional: describing ***what*** a software system should do given a certain condition

Examples:

- Send email when a report is submitted.
- The user should be able to retrieve his/her password by email.

Non-functional: constrain ***how*** a software system will achieve functional requirements.

Examples:

- The software must be fast.
- The software must be secure.

- SRS example: https://buildmedia.readthedocs.org/media/pdf/aakash-tech-support-documentation/latest/aakash-tech-support-documentation.pdf
- SRS typically has no code-specific instructions.
- Customer specifications are rarely code-specific.

Software Design

- It is difficult to generalise software design because it is governed by specific SRS.
- Nevertheless, a fast, memory efficient, and user-friendly software is always better than a software that is not.

Non-functional requirement	What can be done?
Memory efficient	Use dynamic memory as much as possible
Fast	Minimise the complexity of the algorithm
User-friendly	Make the design intuitive

- Typically, a Software Design Description (SDD) is given by software designer to the development team to implement, a common SDD is the IEEE Std 1016-2009 (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5167255&isnumber=5167254)

Software Design - Algorithms

- Algorithms are unambiguous specifications for performing calculation, data processing, automated reasoning, and other tasks.
- Minimising complexity of algorithms is one of the most active research fields in CS.
- The complexity of an algorithm is the amount of resources required for running it.
- The lower the complexity → less computation time → faster software!

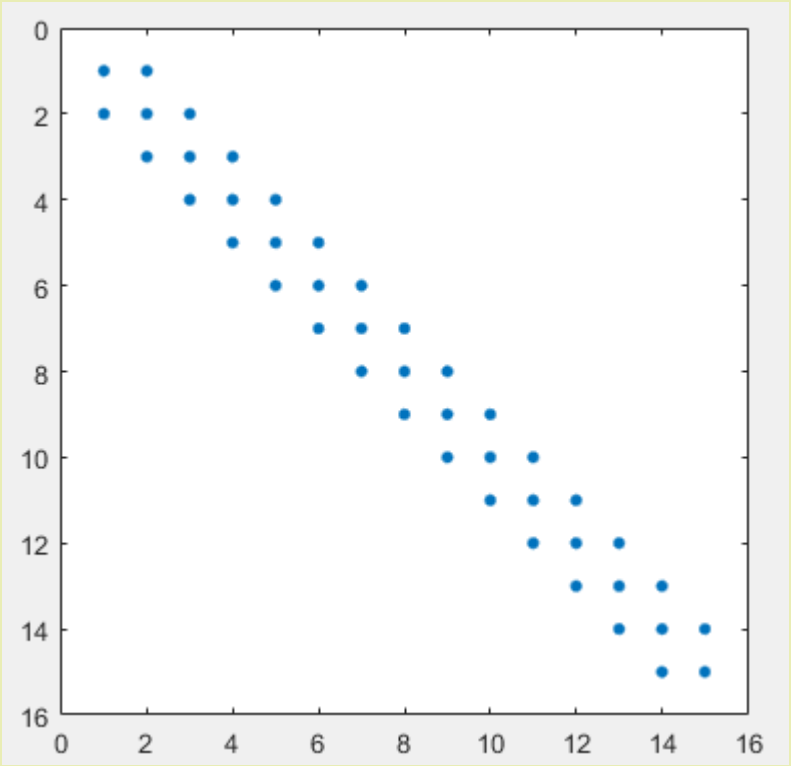
Practical example: Sort algorithm (https://www.youtube.com/watch?v=kPRA0W1kECg)

- Every entry in the `<algorithm>` in STL has a complexity section.

Good practice note: Conduct a bit of research into the best available algorithm for the task you intend to implement in your code and implement it.

Software Design - Algorithms

- Practical example 2: Tri-Diagonal matrices are frequently encountered in scientific computing, where they need inversion to solve a system of equations.
- Three algorithms can be used with different complexities.
- On a teraflops machine, the time taken to invert 10^4 X 10^4 tri-diagonal matrix is:



Algorithm	Complexity	Estimated run time
Cramer's rule	$O(n^4)$	3.44 years
Gaussian elimination	$O(n^3)$	16 minutes
Thomas Algorithm	O(n)	3 seconds

Software Design- Ease of use

- *"The main thing in our design is that we have to make things intuitively obvious,"* the founder and former CEO of Apple, Steve Jobs.
- Using an institutive design of the software/GUI guarantees its ease of use.
- Institutive designs are inspired by users feedback from previous releases, or similarity with well-known software.

Practical example: Linux-GUIs (demonstration).

Good practice note:

- Mimic well-known software such as windows for trivial tasks.
- Deviate from well-known usage to satisfy a software requirement.

Implementation and coding

- Implementation and coding is the main focus of the module.

Good practice note:

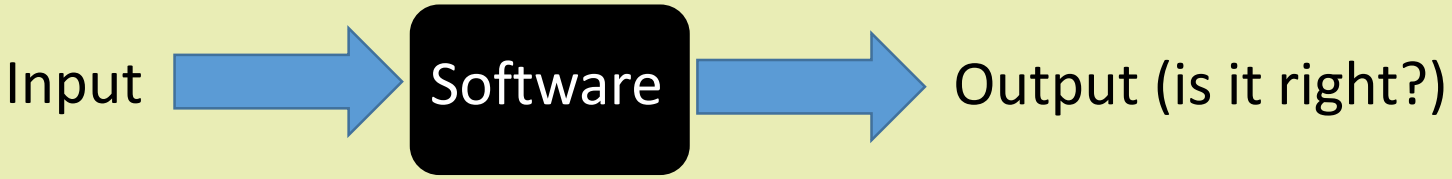
- Make sure your code is well readable, to help your fellow developers to make a good use of your code.
- Always give a preference toward simplicity of the code over its compactness.
- Make sure you stick to any convention in relation to code development when working in a team.

Verification and Validation

- Verification is the process of checking whether the software fulfils its design requirements. (Are we building the product right?)
- Validation is the process of checking whether the software meets the user's expectation. (Are we building the right product?)
- Code-related problems are captured by verification, while functionality is defects are captured by validation tests.
- One common standard of V&V procedure is the IEEE Std 1012-2016 (https://ieeexplore.ieee.org/document/8055462)
- Make sure to mention it in a cover letter if applying to a software validation engineer job.

Verification and Validation

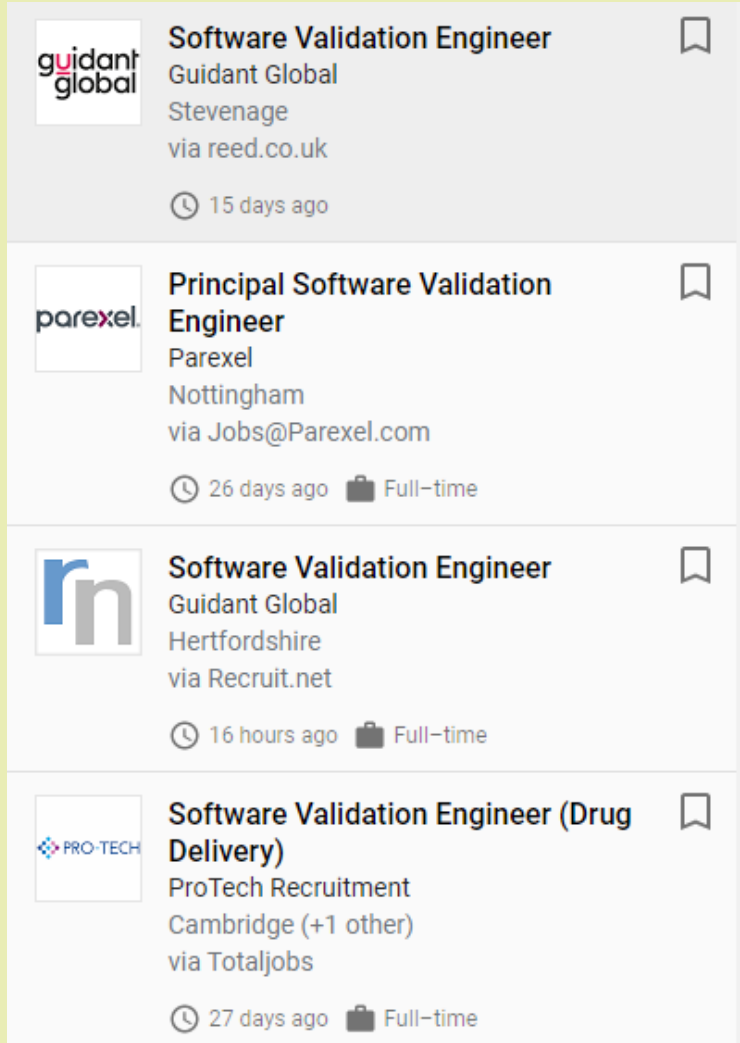
- V&V is commonly done by developers as well as testers.
- Validation tests include black box tests:



- Software testing has a standard as well, Std IEEE 829-2008.

Good practice note:

- Make sure your code works before sharing it with colleagues.
- Benchmark your code against various test cases.
- Consider common user-mistakes in your design.
- Use error-handling to mitigate run-time errors.



Maintenance

- Software Maintenance is the process of modifying a software product after it has been delivered to the customer.
- There are many reasons behind software maintenance:

1. Fix any bugs.
2. Improve the design of the software based on user feedback.
3. If the software is a hardware driver, extending the range of hardware covered.

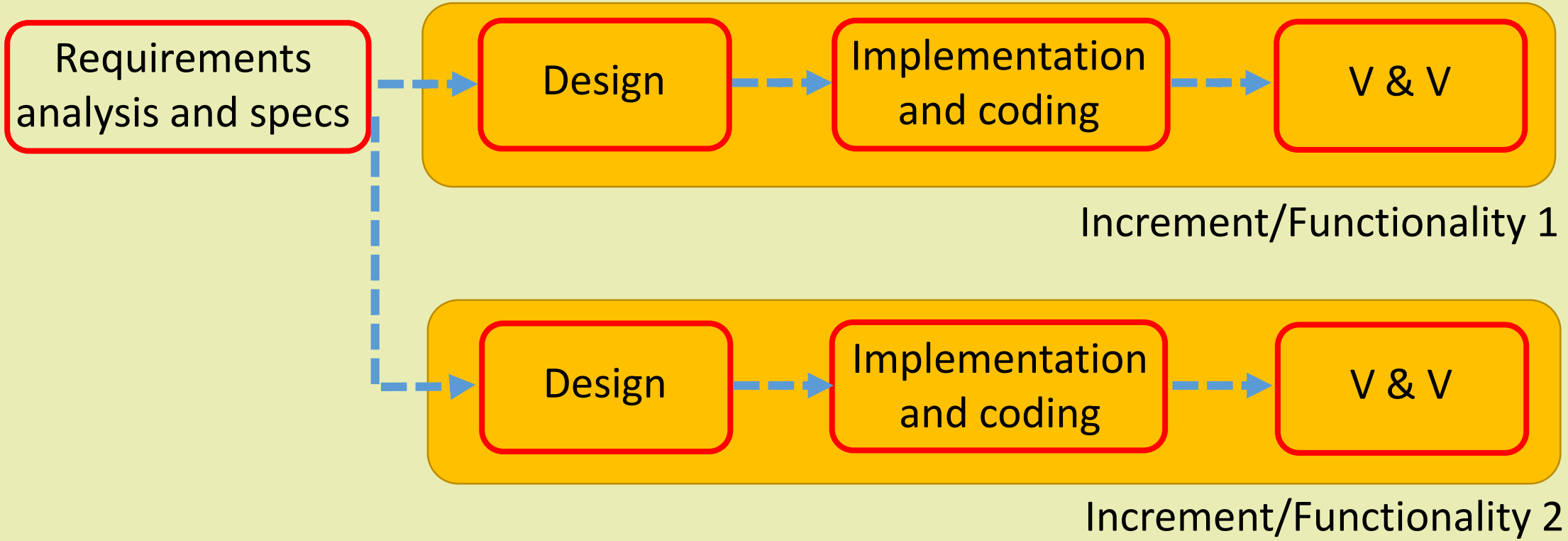
Beyond the waterfall model

- The waterfall model is perfect for education, but too simple for practical use!
- Disadvantages of the waterfall model:

1. No overlapping phases.
2. Difficult to accommodate changes (what if the customer had an extra request?).
3. Slow progress (more people means higher chances of delay).
4. It has relatively high risk (what if there is a problem in one stage?).

- Alternative models include the incremental model, the spiral model, and agile model.

The incremental model



- The incremental model is more flexible, allows overlapping and has lower risk than the waterfall model. So we will stick to it in the rest of the module.

Further information

- Beginning Software Engineering (2015) by Rod Stephens.
- The Pragmatic Programmer (1999) by Andrew Hunt.

Summary

- The basics of software engineering were introduced and discussed.
- The waterfall model was discussed with its advantages and disadvantages.
- The requirement analysis, software design, validation and verification, and maintenance were discussed.
- The incremental model was presented and discussed.