Agile Software Development

COMPSCI5059 - Software Engineering

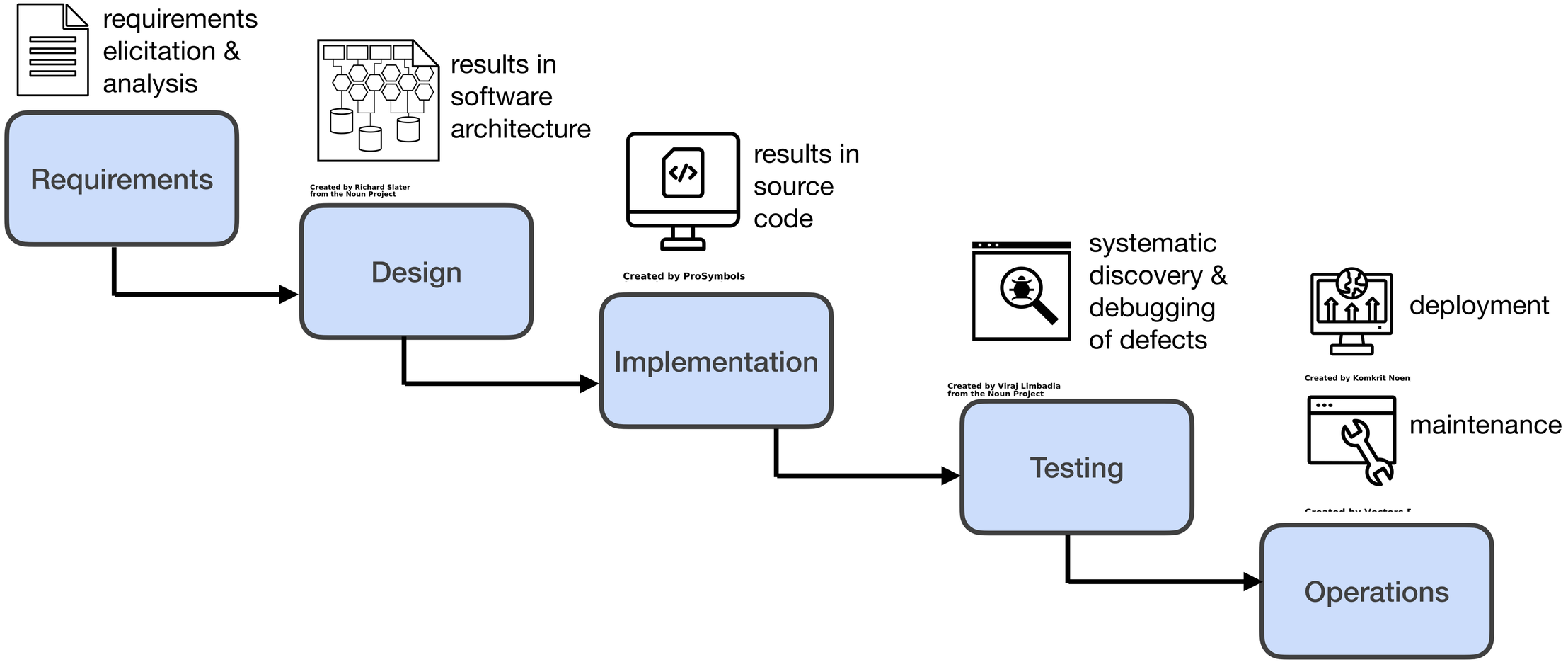
H. Gül Calikli, Ph.D.

Slides are based on the lecture notes by Ron Poet

Overview

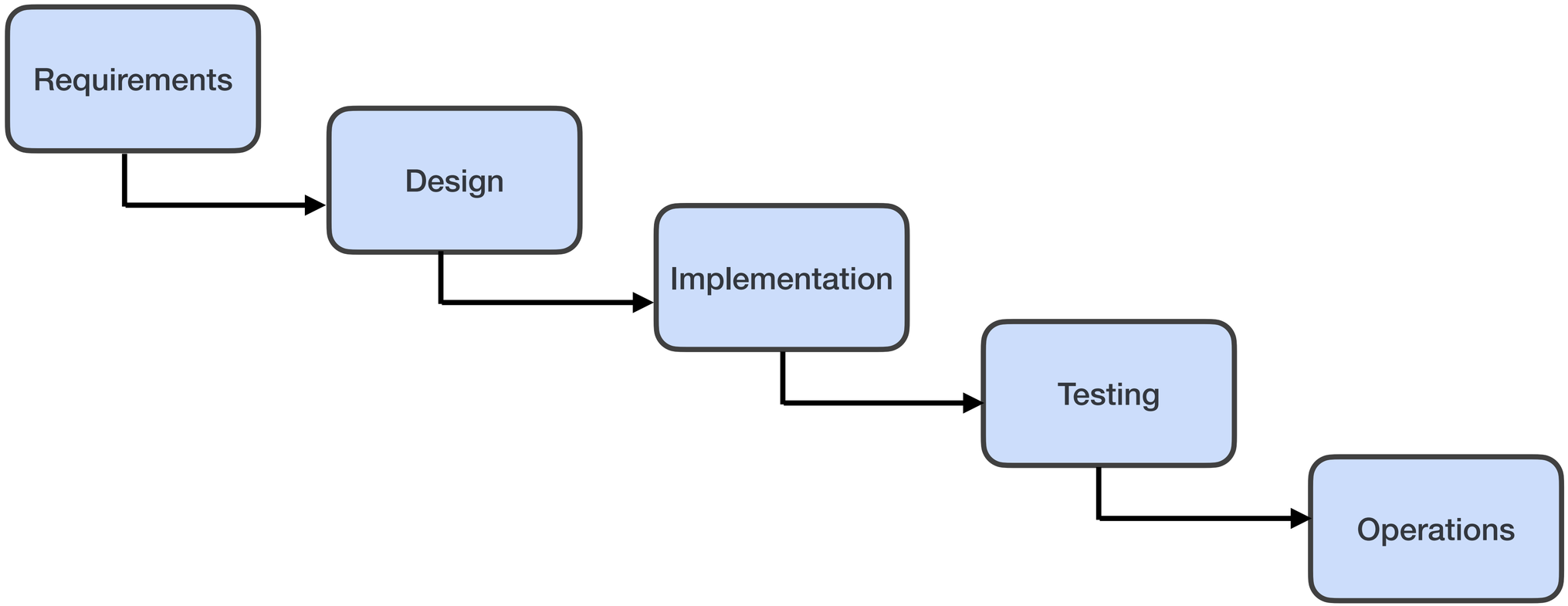
- [ ] The Waterfall Model — old way of developing software
- [ ] Agile Software Development
- [ ] Lean Software Development — a conceptual framework that support agile software development

The Waterfall Model



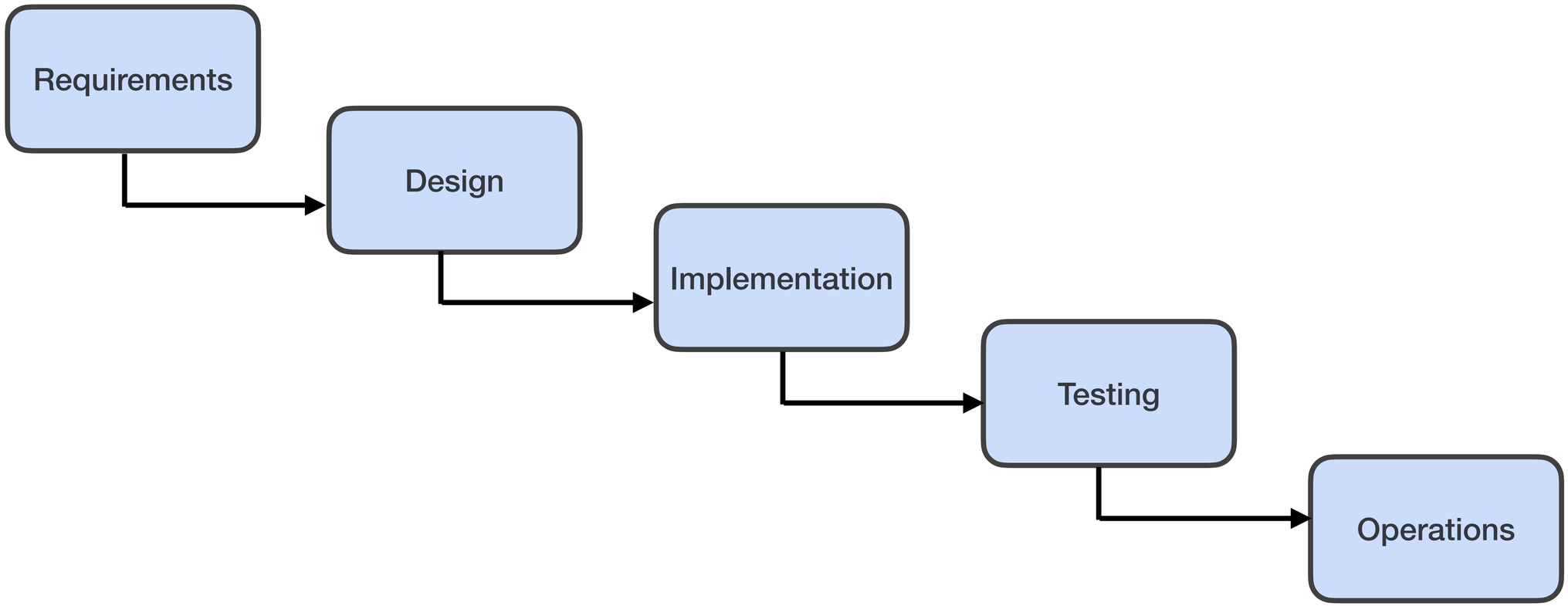
The Waterfall Model

Each step had to be completed before the next step started.



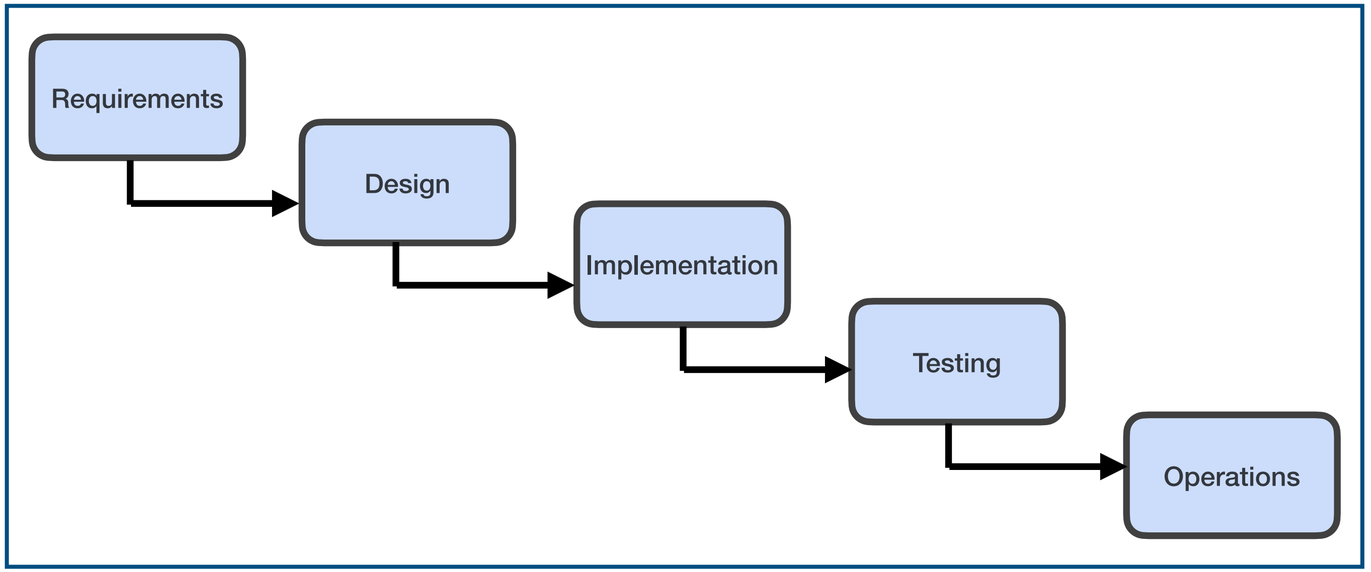
The Waterfall Model

Each step produced a lot of documentation (to record all decisions made).



The Waterfall Model: Problems

- Difficult to keep all documentation consistent & up to date
- Unrealistic to think that all decisions made at the start of the projects are correct - issues became apparent after they were worked on



- Customer did not see the profit for a couple of years - and they realised that this is not what they actually want
- Projects took several years to finish

Agile Software Development

A new approach was needed that was:

- Quicker
- Easier to change as the project progressed
- Provided early customer feedback
- Allowed effective monitoring of progress

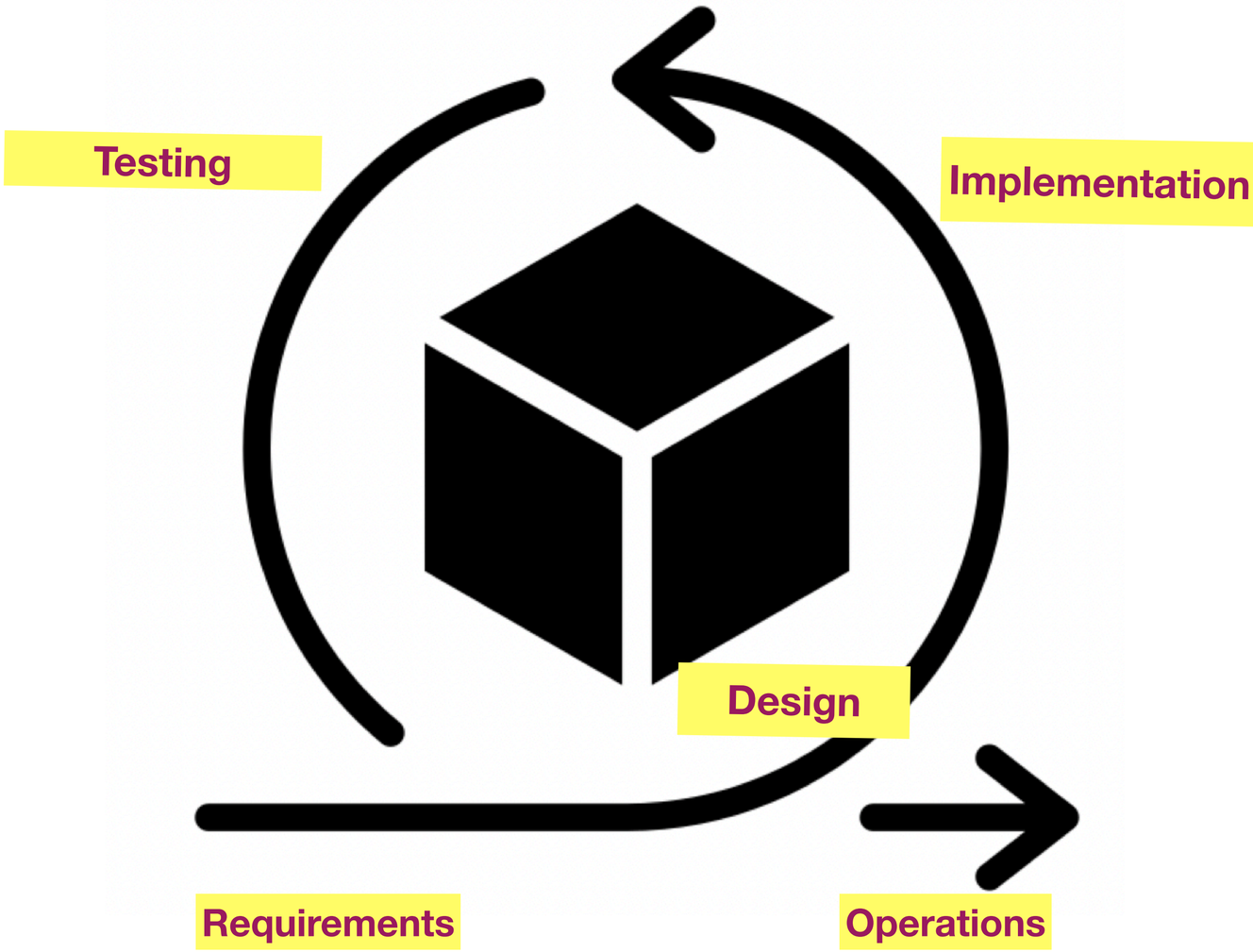
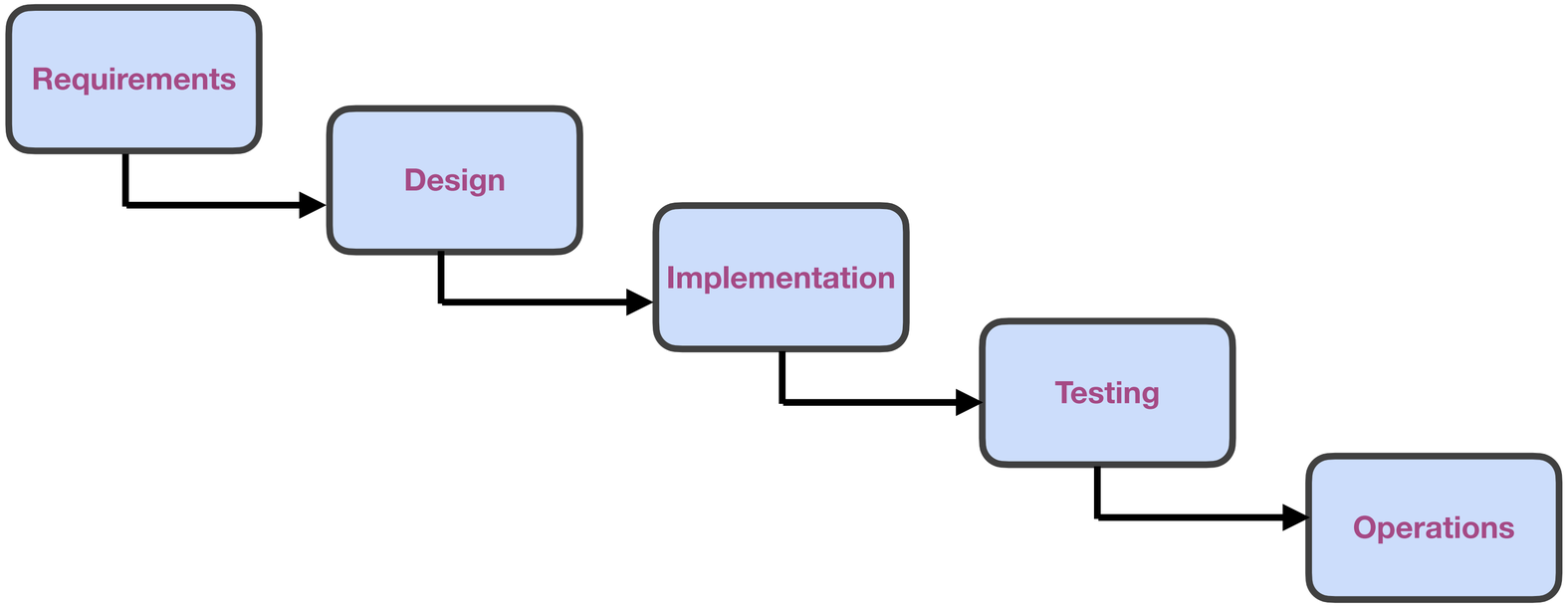


Figure was created by Eko Purnomo from the Noun Project

The Waterfall Model



Agile Software Development

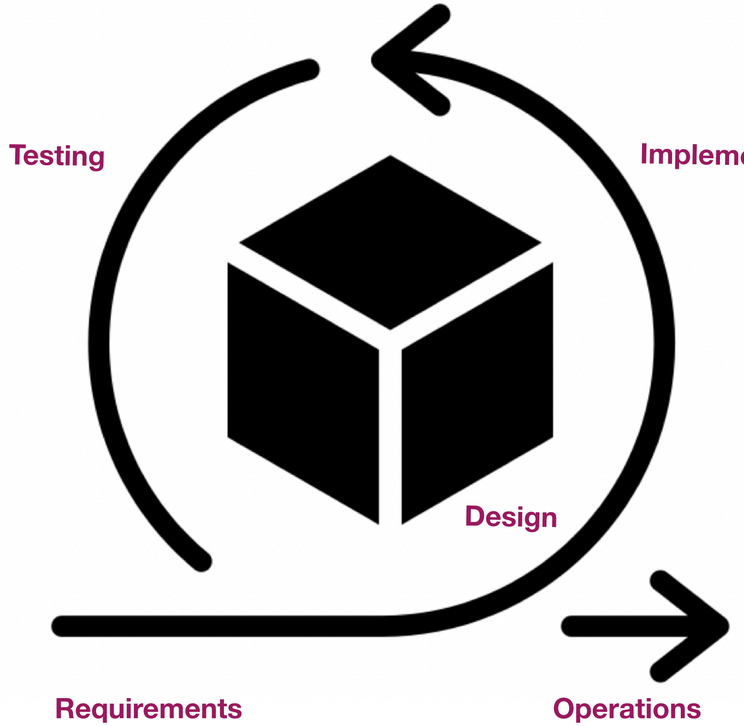


Figure was created by Eko Purnomo from

Overview

- [x] The Waterfall Model — old way of developing software
- [x] Agile Software Development
- [ ] Lean Software Development — a conceptual framework that supports agile software development

Origins in Toyota

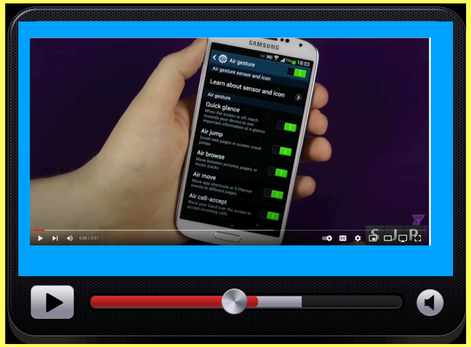
- Lean Development was a new way of designing and building cars.
- Lean Software Development transferred the idea of Toyota's Lean Development to software — The idea became popular around 1990s.

7 Principles of Lean Software Development

1. Eliminate Waste
2. Amplify Learning
3. Decide Late
4. Deliver Fast
5. Empower the Team
6. Build Integrity in
7. See the Whole

Eliminate Waste

A video on the “Air Gesture” feature:
https://www.youtube.com/watch?



dding work. Some examples of waste are:

tation

r of software projects from one team to another

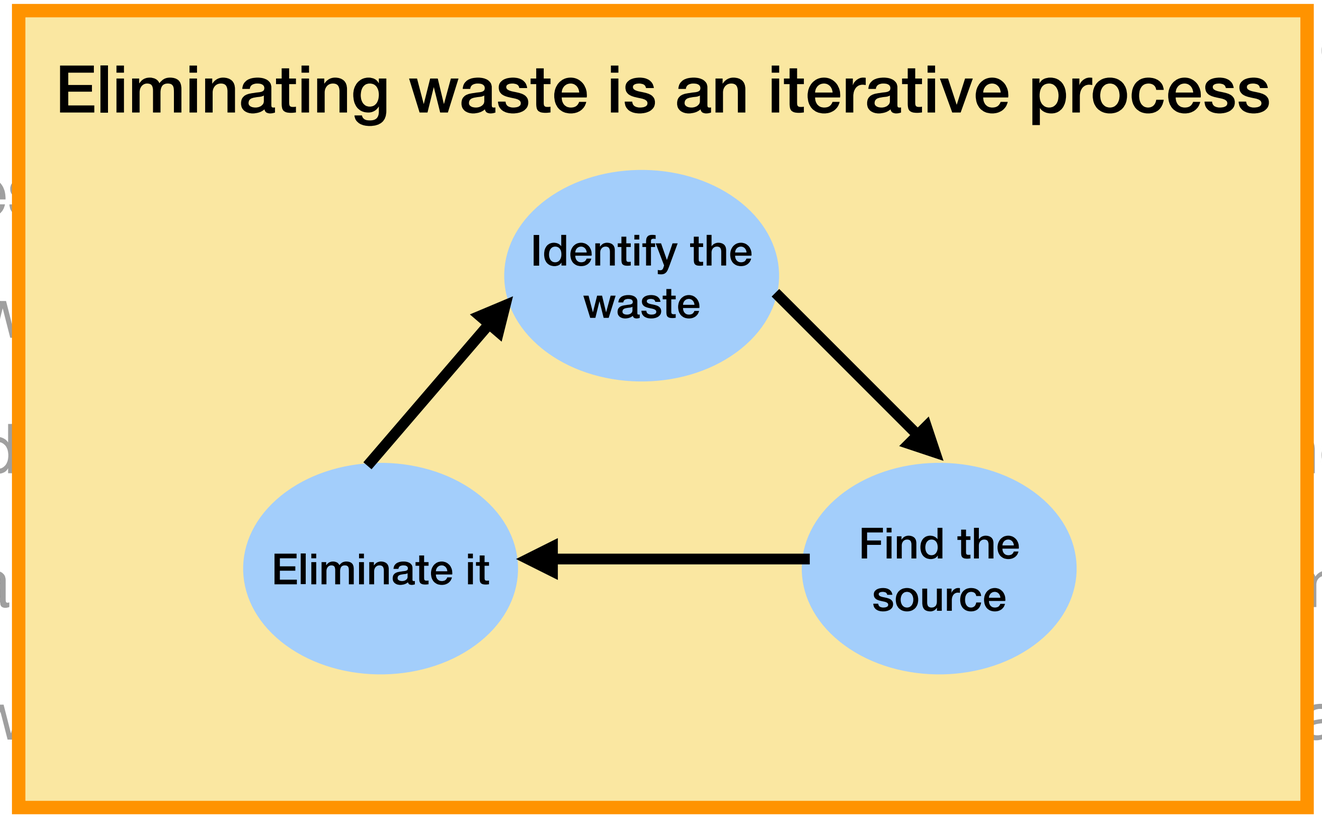
☑ extra features (e.g. “Air Gesture” feature of Samsung Galaxy S4)

Eliminate Waste

- Waste is non-value-adding work. Some examples of waste are:
  - ☑ excess documentation
  - ☑ software defects
  - ☑ hand-offs: transfer of software projects from one team to another
  - ☑ extra features (e.g., “Air Gesture” feature of Samsung Galaxy S4)
  - ☒ throw away code experiments (some look like waste but they are not)

Eliminate Waste

- Waste is
- of waste are:
  - ☑ exce
  - ☑ softw
  - ☑ hand
  - e team to another
  - ☑ extra
  - nsung Galaxy S4)
  - ☒ throw
  - aste but they are not)

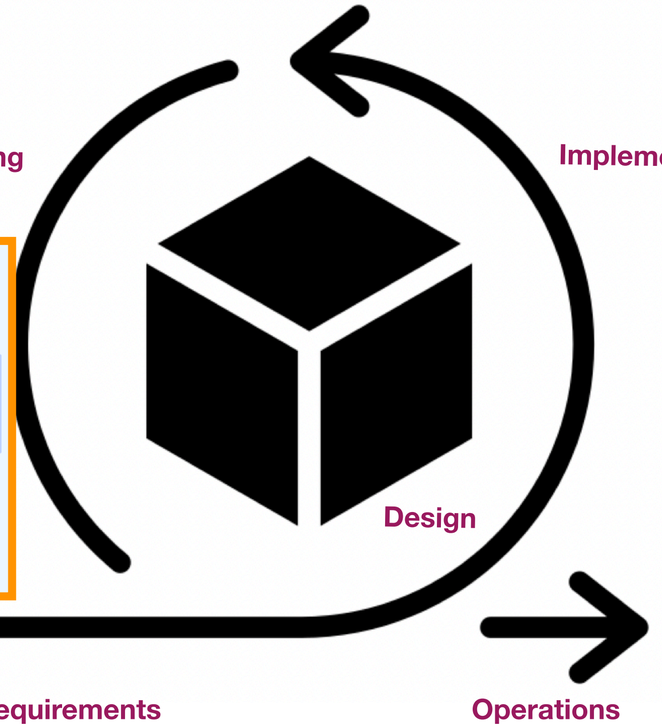


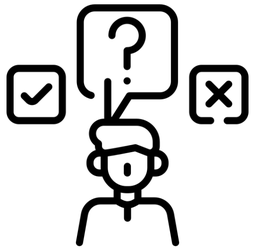
Amplify Learning

- Software development is a continuous learning process.

- Agile processes make sure that everyone shares in the learning process.
  - Short iterations with reviews provide opportunities for reflection and learning.
  - Developing prototypes and showing them to the customer early makes it easier to learn what is really required.

Reviews at the end of each iteration



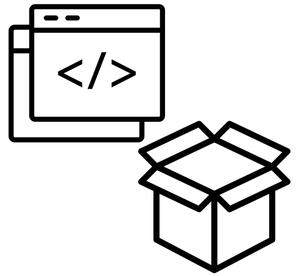


Decide Late

- Keep all options open by putting off decisions as late as possible.

- This can be difficult to get right.
  - It is easy to never make a decision!
  - The iterative approach means that some decisions have to be made at the start of each iteration.

- Hence, plan meetings to consider all the different options.



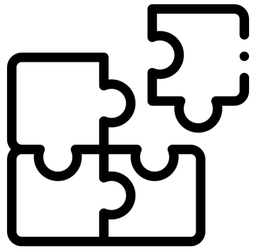
Deliver Fast

- Short iterations produce usable code quickly.
  - It can be difficult to maintain quality as well.

- Use **set-based design** to explore several different options in parallel.
  - Several teams design solutions on the same requirements and the best one is chosen.
  - This may seem like waste as costs might be high.
  - However, not necessarily a waste since the benefits of finding the best solution can be high.
    - Beware that waste occurs when costs are high and benefits are low.

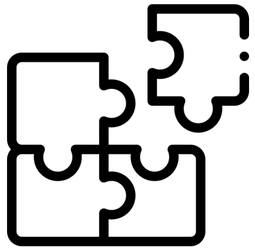
Team Empowerment

- The role of the manager is not to tell people how to do their jobs.
- The managers must help novices improve their skills.
- **Self-organising teams:** The teams should be able to organise themselves without the manager telling them what to do.
- Company procedures should reflect that employees are “individuals,” not interchangeable resources.



In-built Integrity

- **Architectural integrity:** The design is simple and complete at all times
- **Conceptual integrity:** All parts of the system fit well (i.e., not made of separate parts glued together).
  - Small iterations so that requirements are refined together.
  - Regular refactoring of the source code.



In-built Integrity

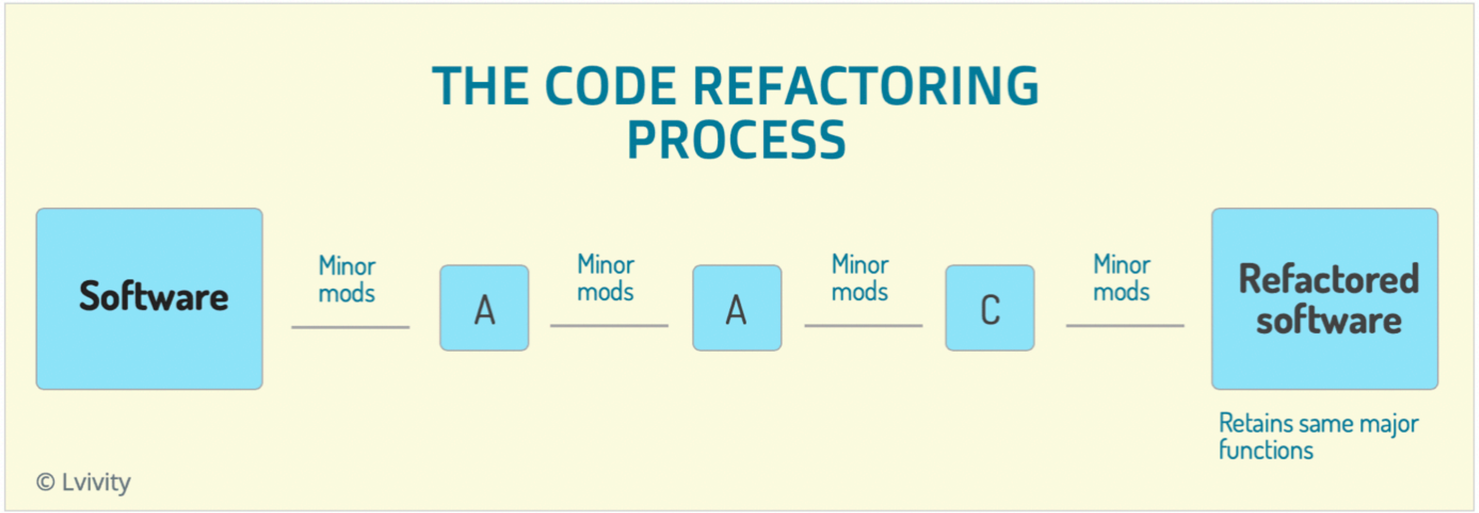
- **Architectural integrity:** The design is simple and complete at all times
- **Conceptual integrity:** All parts of the system fit well (i.e., not made of separate parts glued together).
  - Small iterations so that
  - Regular **refactoring**

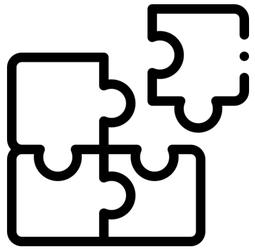
What is refactoring?

Code Refactoring

- The process of restructuring code to improve its internal structure and non-functional features (e.g., maintainability, performance, security).

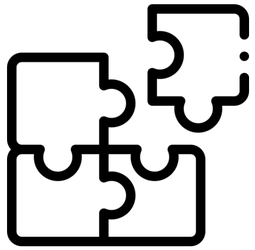
- Code refactoring aims to make the code clean, neat, more efficient and maintainable.





In-built Integrity

- **Architectural integrity:** The design is simple and complete at all times
- **Conceptual integrity:** All parts of the system fit well (i.e., not made of separate parts glued together).
  - Small iterations so that requirements are refined together.
  - Regular refactoring of the source code.
- **Product integrity** through testing.
- **Perceived integrity:** It is not enough to be good, we must also be seen good.



In-built Integrity

- **Architectural integrity:** The design is simple and complete at all times
- **Conceptual integrity:** All parts of the system fit well (i.e., not made of separate parts glued together).
  - Small iterations so that requirements are refined together.
  - Regular refactoring of the source code.
- **Product integrity** through testing.
- **Perceived integrity:** It is not enough to be good, we must also be seen good.

See the Whole

- Everyone involved should see how the final product fits in and will be used (not just focussing on their little piece)
- Everyone has to understand **lean thinking** which consists of:
  - Think big
  - Act small
  - Fail fast
  - Learn rapidly

Interactive Exercise

- Use your computer or smartphone to access the exercise
- If you use your **computer** or **smartphone**: Open a web browser, enter **www.menti.com** and enter code **4862 0159**
- With your smartphone you can also use the following QR code