SFWRENG / MECHTRON 3K04 Software Development

Today: Software Challenges

McMaster University
Engineering
Computing & Software

September 8th, 2025

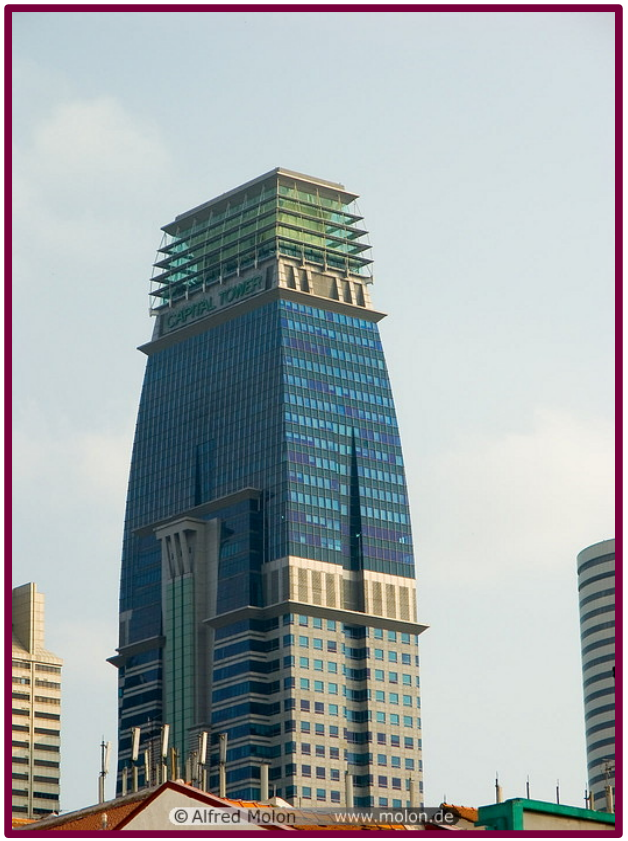
Dr. Angela Zavaleta Bernuy

Software Challenges

- Complexity
- Lack of continuity
- Software does not age in the same way that hardware does
- In general, it is difficult to automate tasks
  - Which is what we do all the time when creating software solutions
- It is easy to write software programs
  - But, it is difficult to get them to be correct, safe, secure and reliable.

Complexity

Three structures of varying complexity:



Complexity: How does it affect us?

Could you build this dog house without instructions?

With some prior experience, I may be able to build this without knowing any requirements or following a design.

Complexity: How does it affect us?

Could you build this house without instructions? (or without knowing the requirements or following a design?)

Maybe … **MAYBE** … we can build this house without requirements, but without a design plan? Or at least blueprint?

Complexity: How does it affect us?

Could you build this building without instructions? (or without knowing the requirements or following a design?)

NO

Without documentation, we cannot really maintain this building.

Complexity: How does it affect us?

The same is true for software!

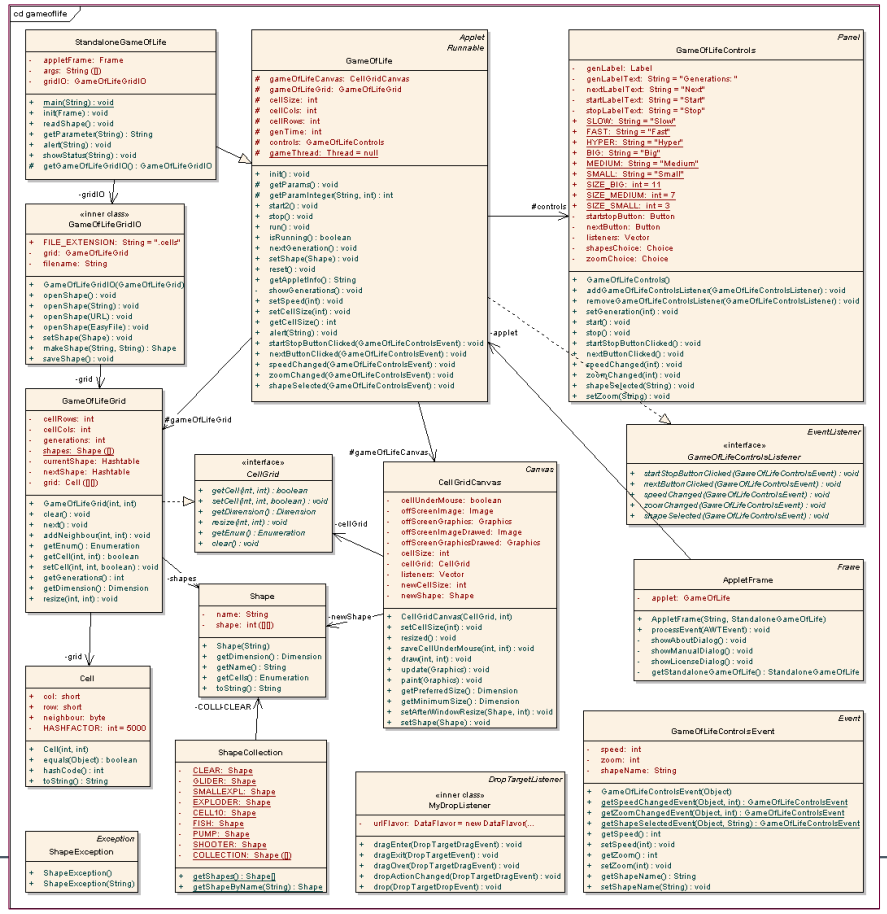
- We accept that we need to *engineer* these structures, but many people resist *engineering* software applications
- The complexity of a physical structure, is at least comparable to a software application.

But how do we measure complexity?

Complexity: Measuring it in Software

Example: UML class diagram.

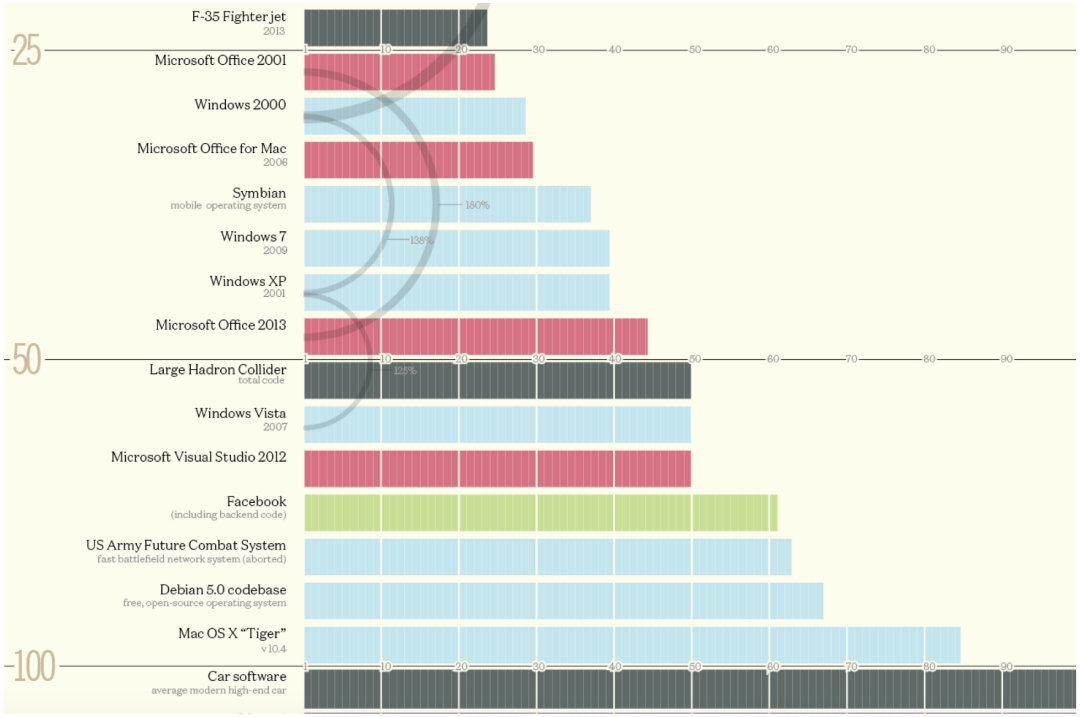
- Each class has a list of attributes (variables, fields) and a list of methods (operations, functions)
- The diagram also show the relationships between classes such as dependencies or inheritance.
- Each method will consist of a number of lines of code.

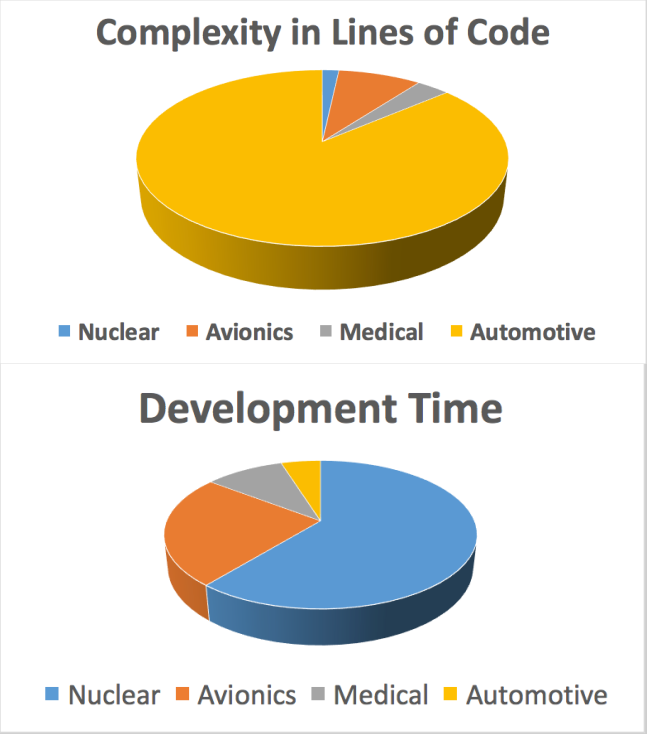


https://bitstorm.org/gameoflife/code/gameoflife-uml.png

Complexity: Lines of Code

Lines of code may not be a great measure of complexity, but they do tell us something about complexity!

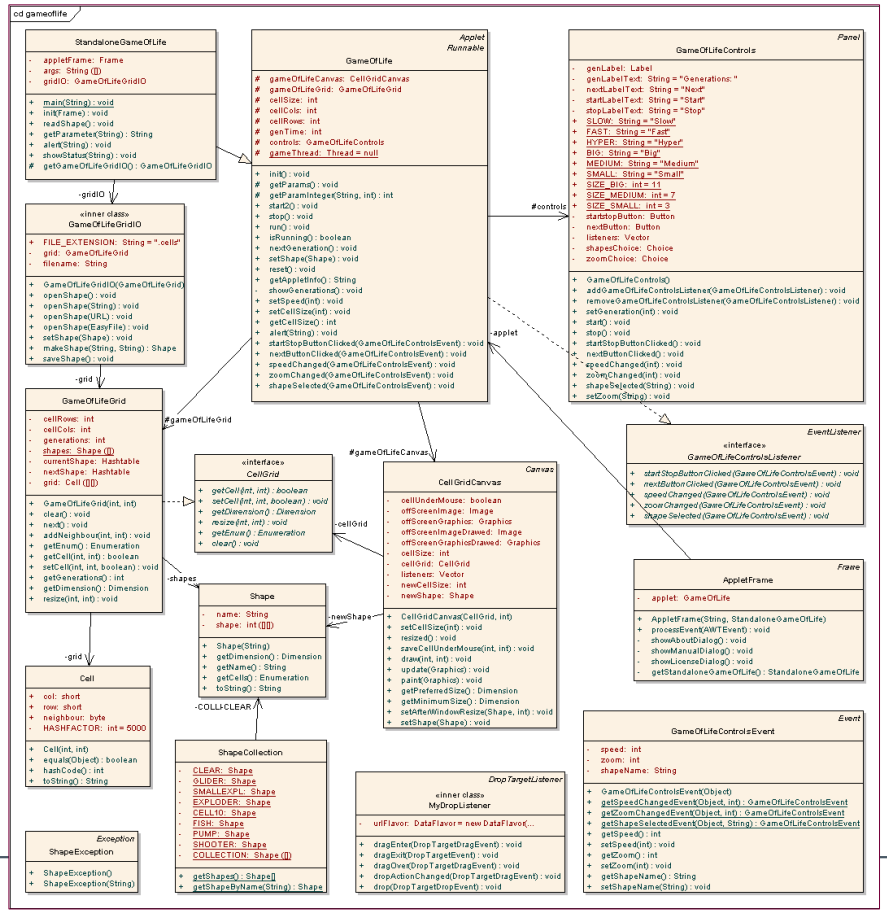




Complexity: Cohesion and Coupling

Example: UML class diagram.

- ***Cohesion*** is determined with how much information is held within a class/component/module (self-contained).
- ***Coupling*** is determined with how much a class/component/module relies on other parts to complete its functionality.

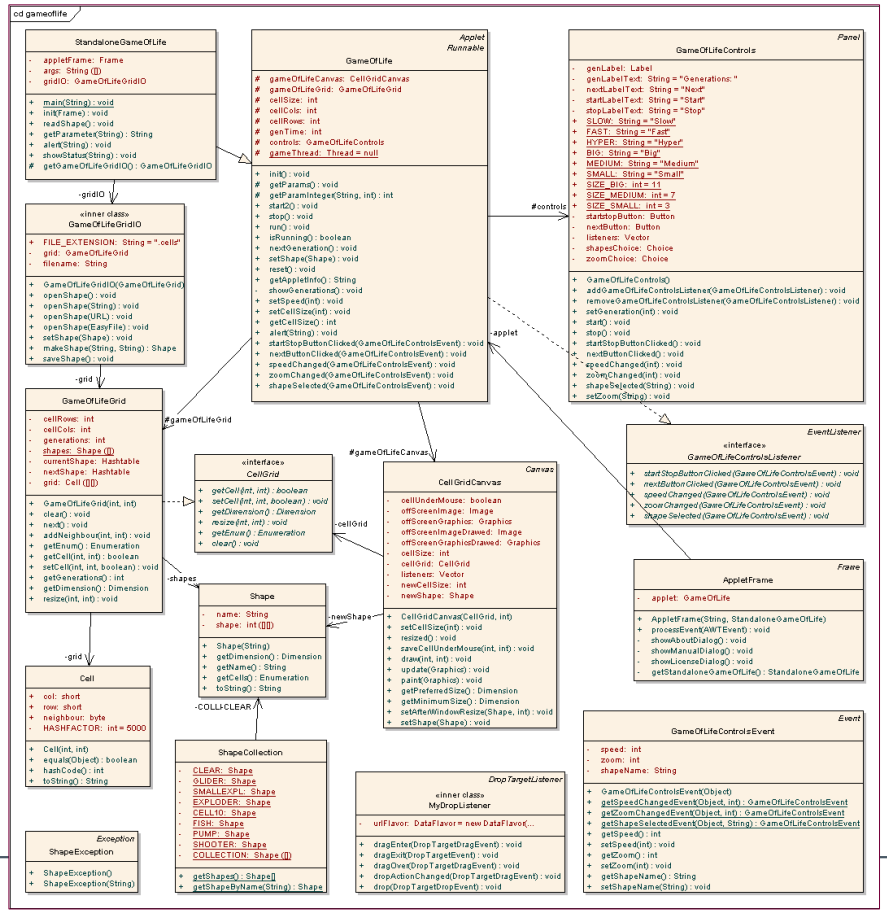


https://bitstorm.org/gameoflife/code/gameoflife-uml.png

Complexity: Cohesion and Coupling

Example: UML class diagram.

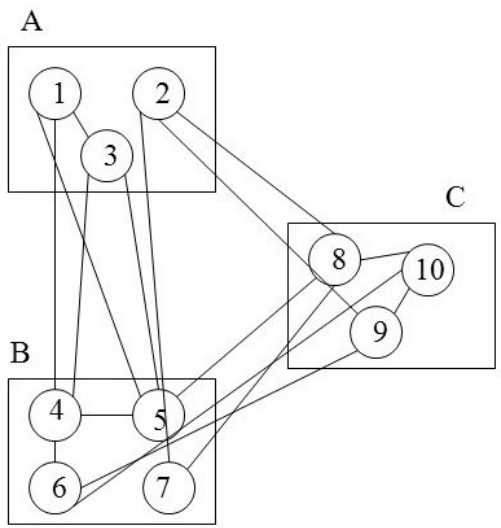
- Each module, should be ***highly cohesive***
  - Module understandable as a meaningful unit
  - Components of a module are closely related to one another
- Modules should exhibit ***low coupling***
  - Modules have low interactions with others
  - Understandable separately
- We need good modularization ☺



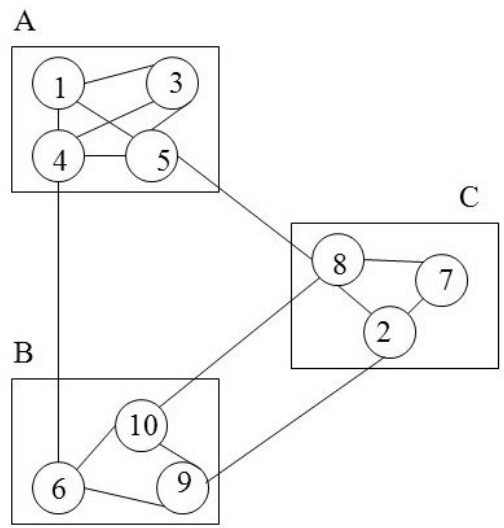
https://bitstorm.org/gameoflife/code/gameoflife-uml.png

Complexity: Cohesion and Coupling – Visually

We can evaluate ***coupling*** from an external view ("uses"), but we need to see internal details to judge ***cohesion***.



Bad modularization:
low cohesion, high coupling



Good modularization:
high cohesion, low coupling

https://devopedia.org/cohesion-vs-coupling

Complexity: Cohesion and Coupling – Match!

A

B

Which of these scenarios represent HIGH cohesion?

Complexity: Cohesion and Coupling – Match!

A

B

Which of these scenarios represent LOW coupling?

Continuity Property

- Most physical products, exhibit a continuity property. That is, an artifact behaves in a continuous, seamless, and unbroken behaviour.
- A small input/change, should result in a predictable small/change in output.
- E.g., a shovel

Continuity Property: Binary Search Example

Table
163
2
-45
3
16
9

Key	3	T
Key	20	F

```python
def binary_search(key, table):
    bottom = 0
    top = len(table) - 1

    while bottom < top:
        if (bottom + top) % 2 == 1:
            middle = (bottom + top - 1) // 2
        else:
            middle = (bottom + top) // 2

        if key <= table[middle]:
            top = middle
        else:
            bottom = middle + 1

    found = (key == table[top])
    return found
```

What if we decide to remove this else block?

Does Software Age?

- Software doesn't really behave different as it gets older.
- Right?!
  - Sort of!
- Software runs on hardware platforms.
  - The platform can change: it can age and malfunction, but it also can be much faster!
- Malfunctions and speed changes have implications for the way that the software behaves.
- If we do our job right, we can plan for those changes so that the software does not "age"

Takeaways!

- The engineering world has found that software is versatile
- Difficult and complex functions are easier to do in software than in hardware.
  - We need good modularity! **High cohesion and low coupling.**
- For centuries, we have learned to build system and test them relying on the fact that physical systems exhibit a continuity property, **but software systems do not!**
- Software does not age in the way that hardware does, so people disregarded this as a problem
  - However, software platforms (and the environment in which the software operates) **do age** and this does affect the way the software behaves!

SFWRENG / MECHTRON 3K04 Software Development

Today: Software Development Styles

McMaster University
Engineering
Computing & Software

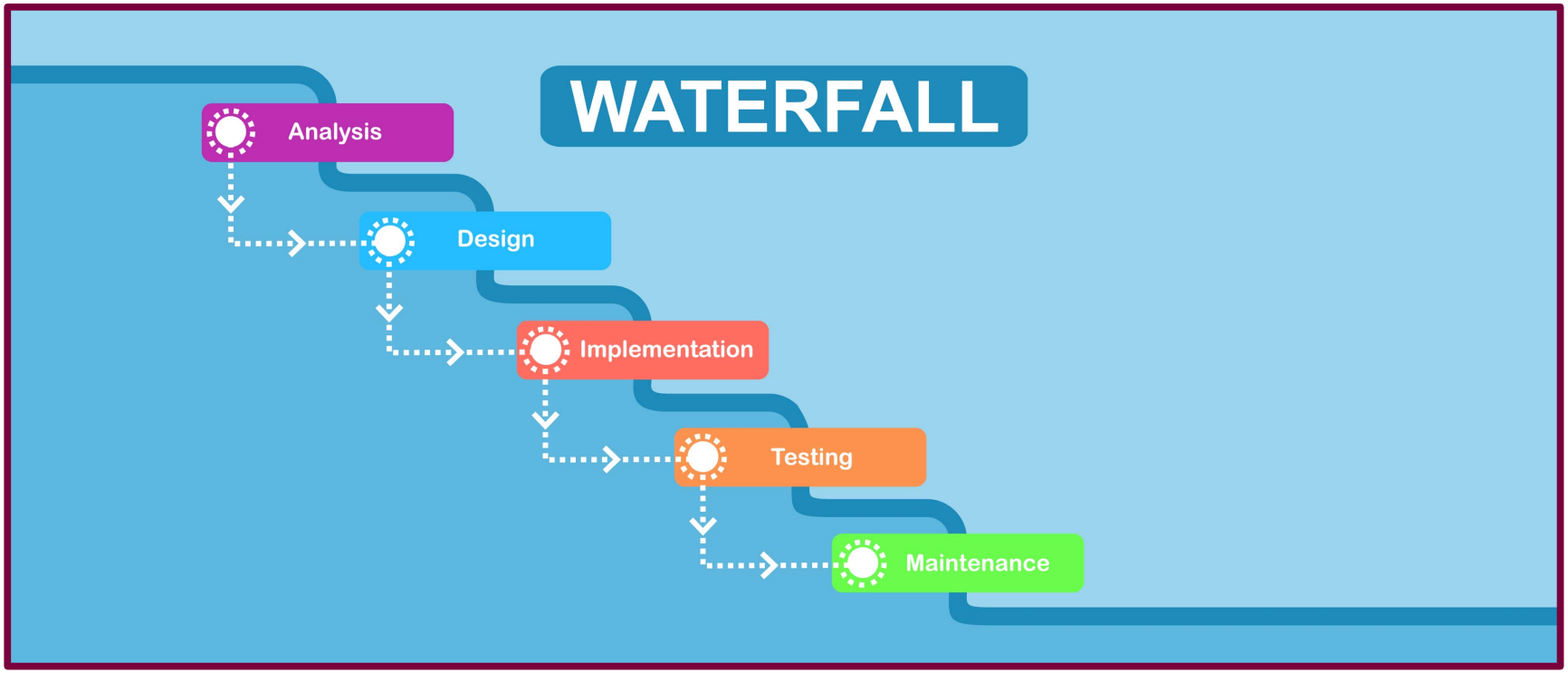
September 10/12th, 2025

Dr. Angela Zavaleta Bernuy

Software Development Styles

- How do you **approach** software development?
- How do you **scope** software?
- How do you **support** software?
- How do you know when **deliver** software?
- Today: Let's look at the many Software Development Life Cycle (SDLC) methodologies / models!

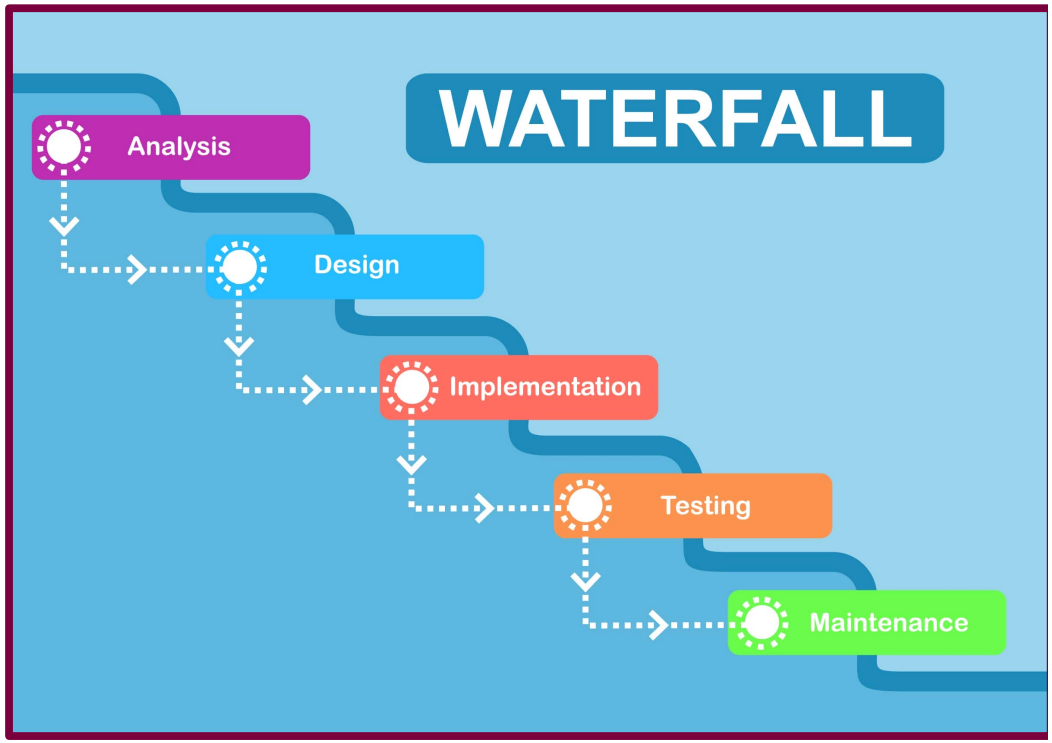
(Traditional) SDLC: Waterfall Model



https://www.chetu.com/blogs/technical-perspectives/understanding-software-development-life-cycle.php

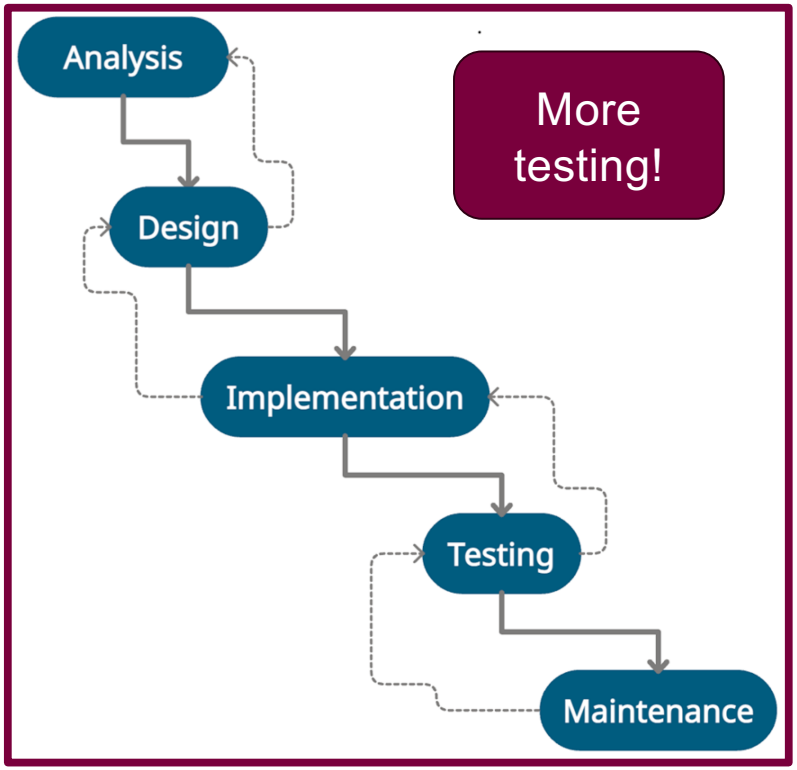
(Traditional) SDLC: Waterfall

- How do you **approach** software development?
  - Development is supported in sequential phases
  - Each phase must be completed before moving to the next
- How do you **scope** software?
  - In the requirements phase
  - A formal requirements baseline is set
- How do you **support** software?
  - In the maintenance phase
- How do you know when **deliver** software?
  - All requirements have been met and verified
  - Delivered in a single, final release



https://www.chetu.com/blogs/technical-perspectives/understanding-software-development-life-cycle.php

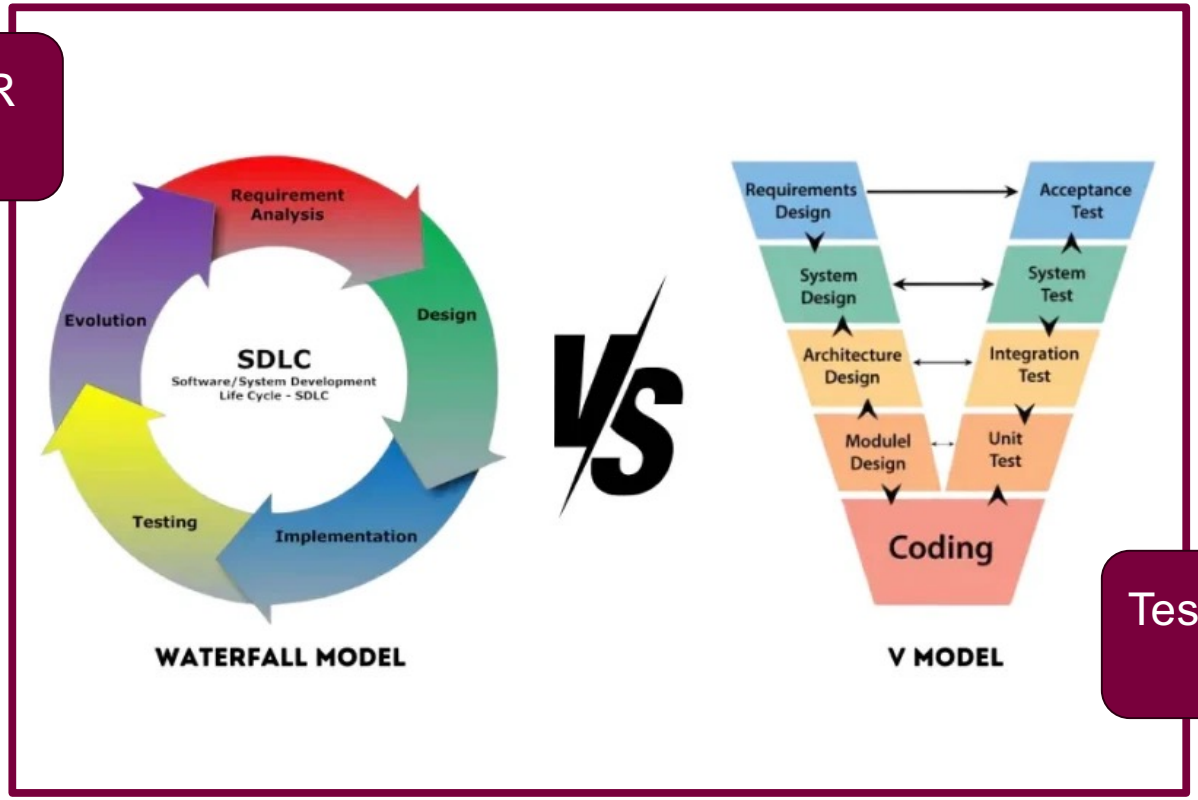
(Traditional) SDLC: Modified Waterfall



https://www.mdpi.com/2571-5577/6/6/108

(Traditional) SDLC: V Model

Testing AFTER development

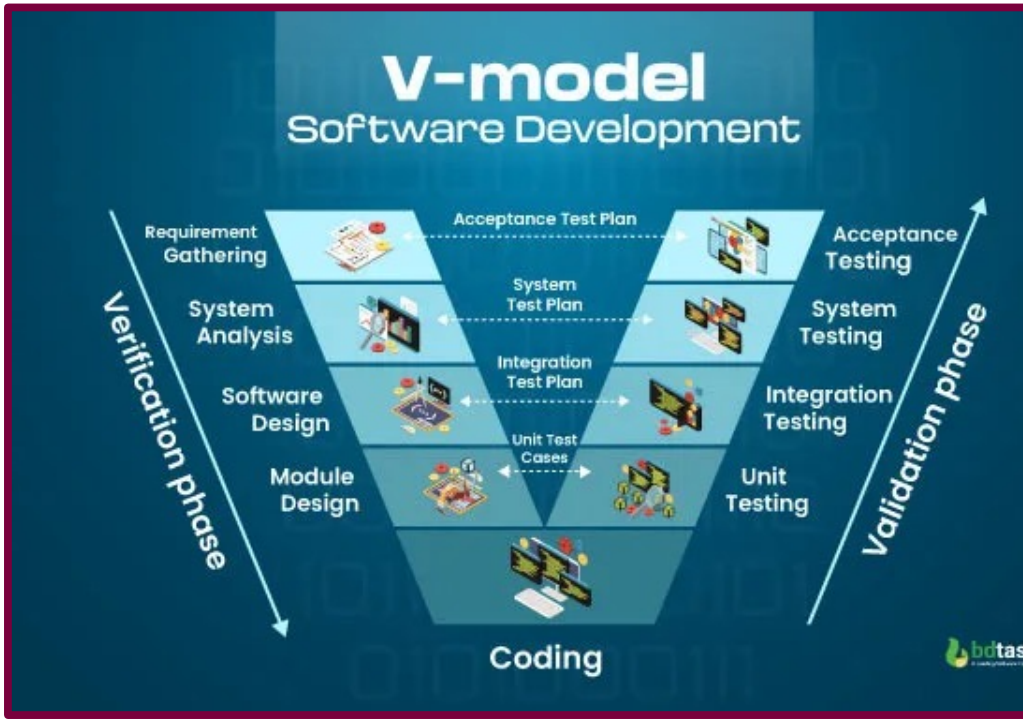


Testing ALONGSIDE development

https://stepmediasoftware.com/blog/v-model-and-waterfall-model/

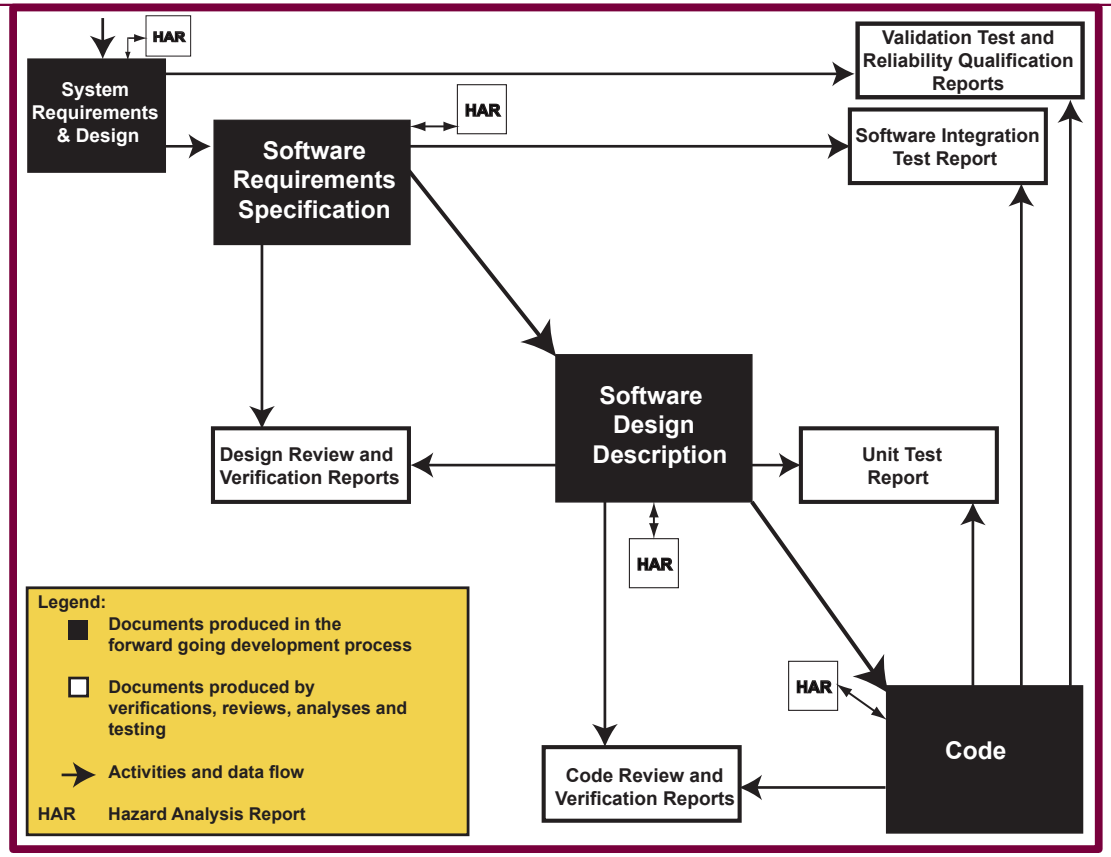
(Traditional) SDLC: V Model

- How do you **approach** software development?
  - Development is supported in sequential phases
  - Testing activities are planned in parallel
- How do you **scope** software?
  - In the requirements phase
  - A formal requirements baseline is set
- How do you **support** software?
  - In the maintenance phase
- How do you know when **deliver** software?
  - All requirements have been met and verified
  - Delivered in a single, final release



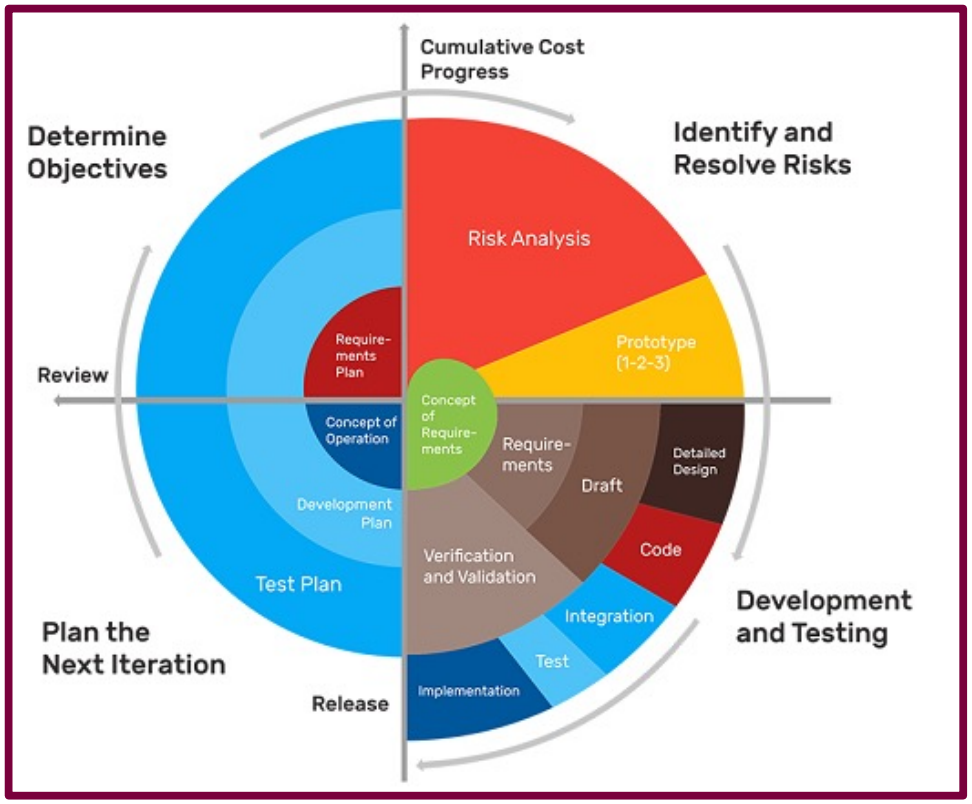
https://www.bdtask.com/blog/v-model-in-software-development

(Traditional) SDLC: Assurance in Critical Env.



https://www.cl.cam.ac.uk/archive/mjcg/FMStandardsWorkshop/talks/MLawford.pdf

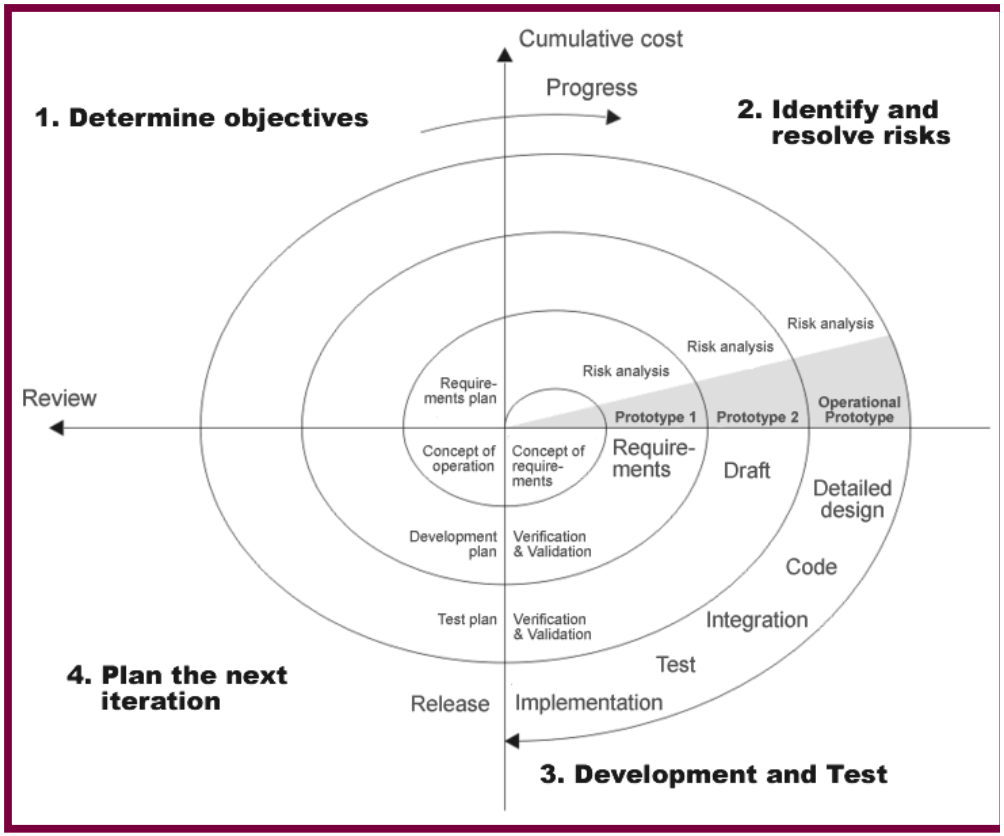
(Traditional) SDLC: Spiral Model



https://www.sovereignconsult.com/blog/spiral-model-methodology/.

(Traditional) SDLC: Spiral Model

- How do you **approach** software development?
  - Development is supported in iterative cycles
  - Planning, risk analysis, engineering, and evaluation
- How do you **scope** software?
  - Scope is defined incrementally at every cycle
  - Adapt to changing needs!
- How do you **support** software?
  - In the maintenance phase, but also during development
- How do you know when **deliver** software?
  - It can be incremental!



https://www.bdtask.com/blog/v-model-in-software-development

Let's get into the timeline…

1970s
Waterfall

- Sequential, plan-driven

→

1980s
V Model

- Verification and validation at every stage

→

Mid-1980s
Spiral Model

- Risk-driven iteration

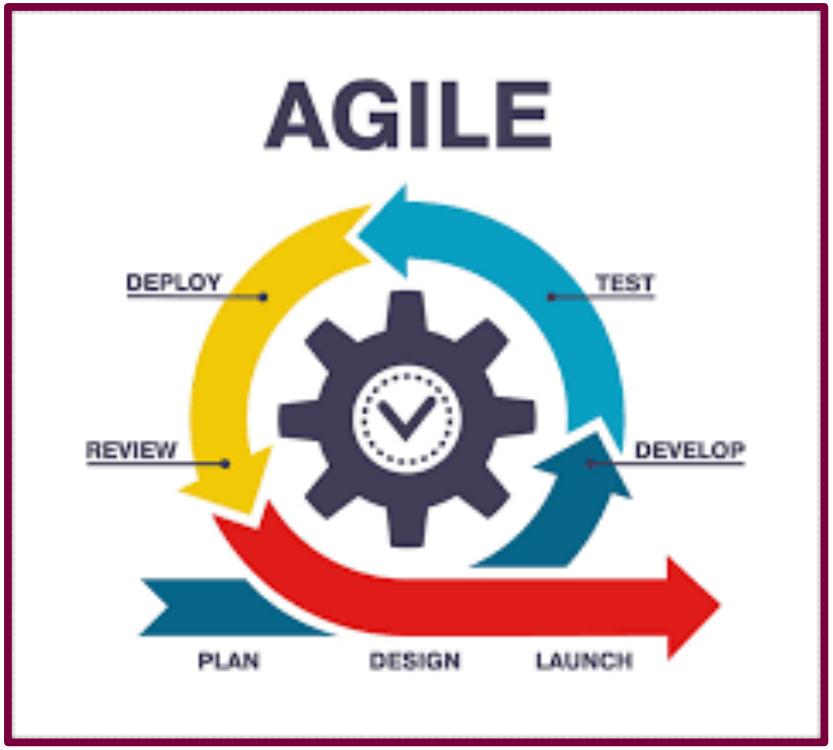
→

2000s

- We need **FASTER** results

SDLC: Agile

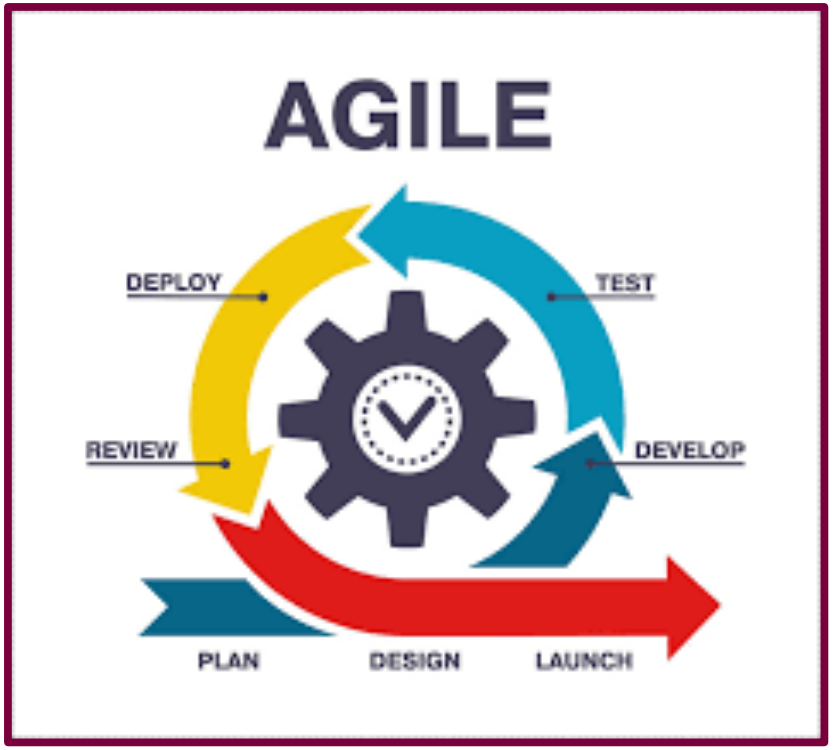
- Iterative and flexible approach that breaks projects into small, manageable cycles (*iterations*)
- More suitable to address changes in software development
- Incremental development, successive release of products
- Follow the Agile Manifesto principles: https://agilemanifesto.org/
- Different versions:
  - SCRUM
  - Extreme Programming (XP)
  - Kanban
  - Etc.



https://yuktisolutions.com/blog/agile-sdlc-for-your-project

SDLC: Agile

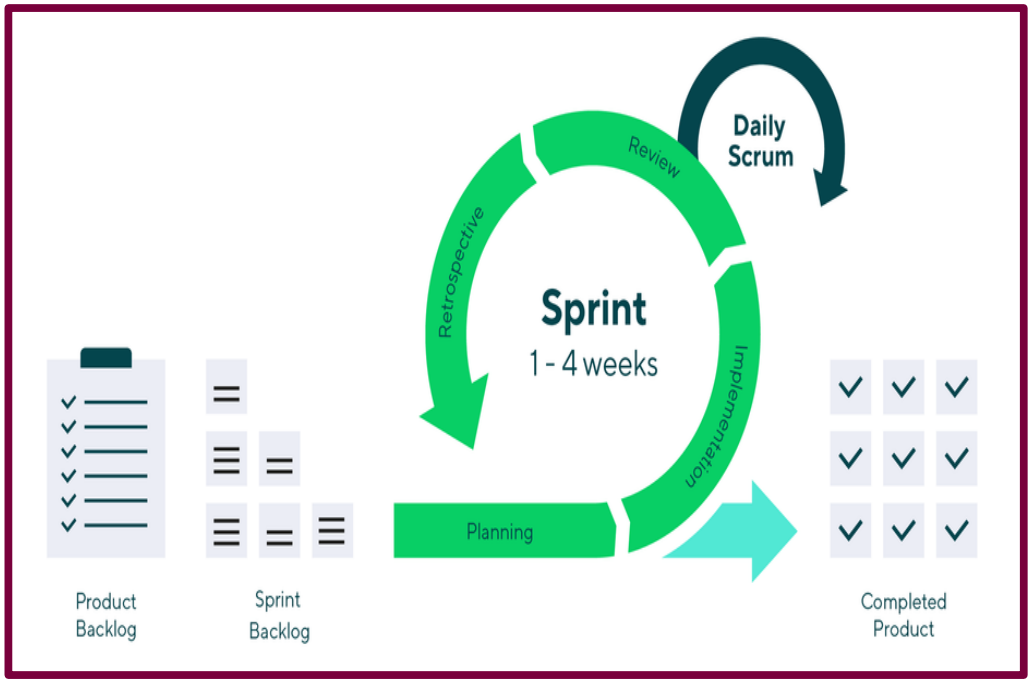
- How do you **approach** software development?
  - Iterative and incremental
  - Work is divided into short cycles (iterations)
- How do you **scope** software?
  - Managed in a prioritized backlog of features or user stories
  - Requirements are not all fixed upfront
- How do you **support** software?
  - Continuous support and ongoing improvement
- How do you know when **deliver** software?
  - Early and often
  - Working, tested and valuable



https://yuktisolutions.com/blog/agile-sdlc-for-your-project

SDLC: Agile – SCRUM

- Origin of the word: rugby
  - Players huddle and work together to move the ball forward
  - Which is what Agile teams should do!
- Users involved:
  - Product owner
  - Scrum master
  - Other members
- Workflow:

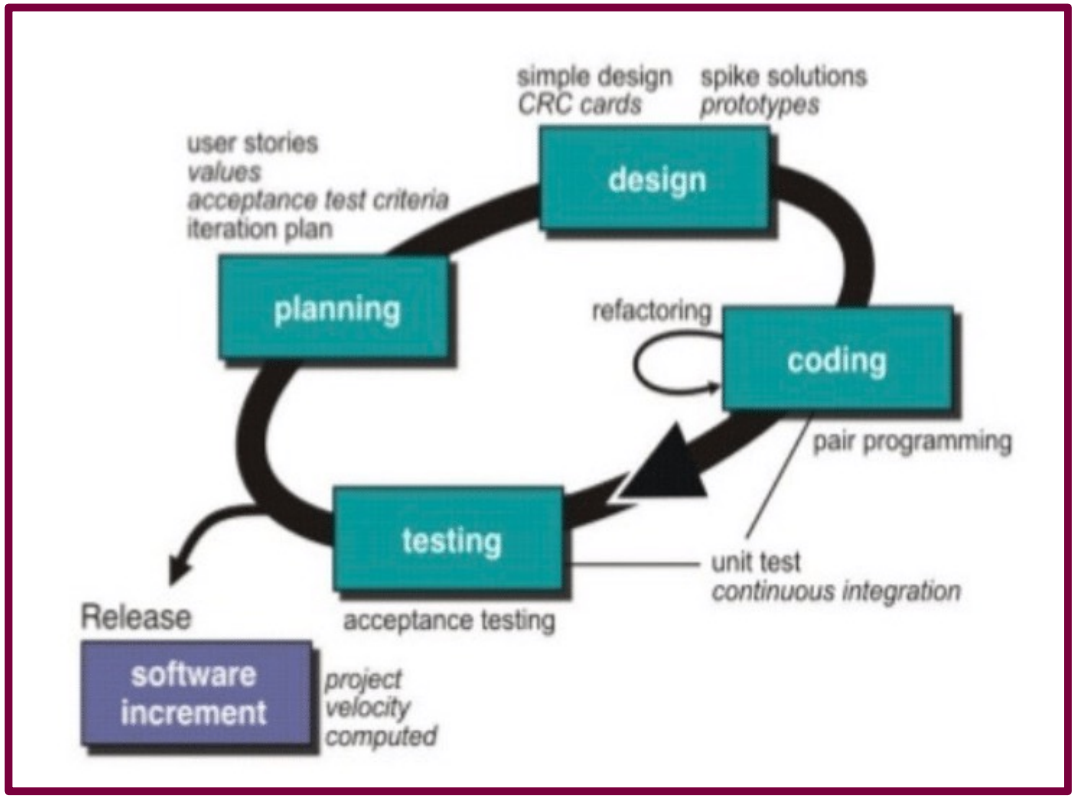


https://www.wrike.com/agile-guide/agile-development-life-cycle/

SDLC: Agile – Extreme Programming (XP)

- XP considers four basic activities: coding, testing, listening and designing
- Coding: "*Without code, there is no working product*"
  - Practices: pair programing, code review
- Testing: Test-driven development, continuous testing
- Listening:
  - Planning game: short planning session each iteration, typically once a week
  - Release planning: set goals for near-term releases
  - Iteration planning: details tasks for developers during the next short cycle
- Designing
  - Simple design: build the simplest system that works, then improve it continuously
  - Refactoring

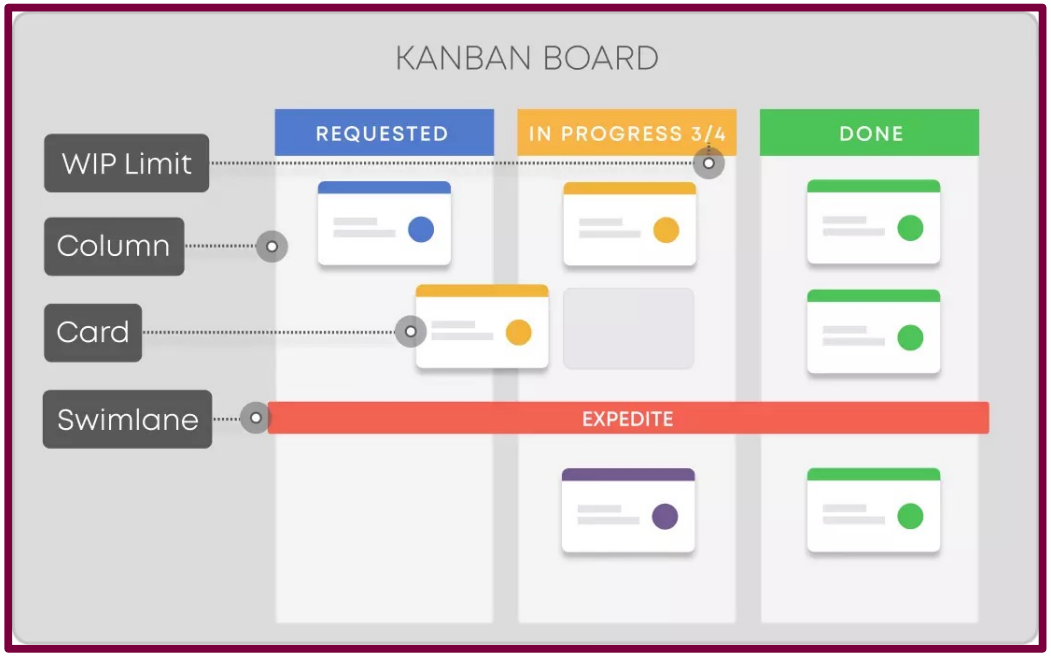
SDLC: Agile – Extreme Programming (XP)



https://www.researchgate.net/figure/Extreme-Programming-XP_fig2_356185795

SDLC: Agile – Kanban

- ”Visual card” in Japanese
- Focuses on visualize work, limit work in progress and improve flow
  - Visualizing work: Kanban board
  - Only allow a small number of tasks per column
  - Track how smoothly tasks move across
- Make process policies explicit
- Continuous improvement



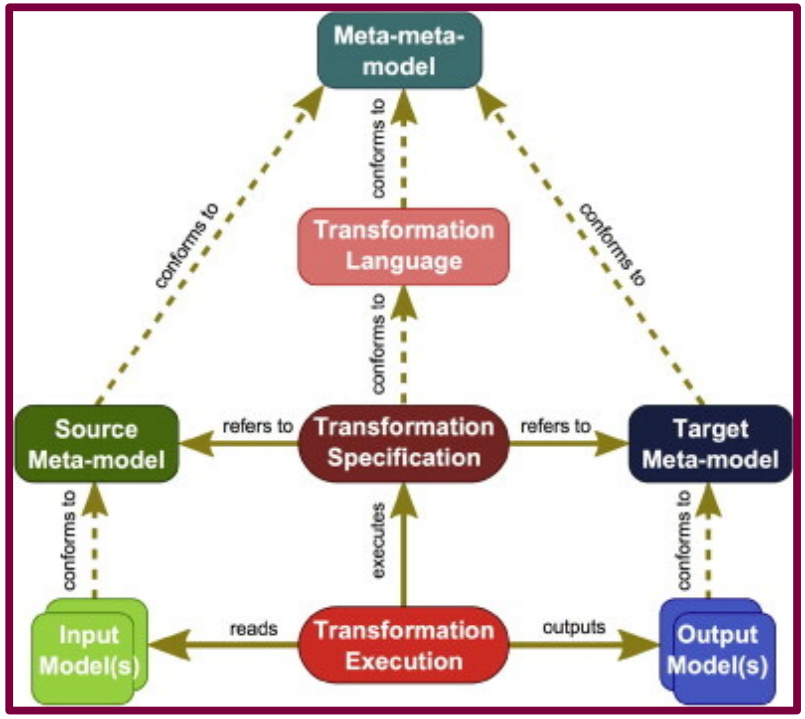
https://businessmap.io/kanban-resources/getting-started/what-is-kanban-board

Modern Trends in SDLC

- Agile:
  - Meant for smaller teams and smaller development cycles
  - Some people are distrustful of how Agile can be used to handle safety critical engineering
- Waterfall
  - Meant for bigger teams and bigger projects
  - Some people dislike how long it takes to develop using waterfall
  - Tried and tested for development of critical components
- We need a methodology that can fit in the SDLC to make development of safety critical systems efficient
  - Model-Driven Engineering!

Model-Driven Engineering (MDE)

- It makes models the main artifact of development instead of raw code
  - Code can be generated automatically from these models
  - E.g., we can transform a UML diagram into Java classes, or a state machine model into embedded system code ;)
- Goal: raise the level of abstraction so developers think in **domain concepts** rather than low-level programming
- Reduce manual coding and relies on code generation and automation tools
- **Is MDE an SDLC?**



https://www.sciencedirect.com/topics/computer-science/model-driven-engineering

MDE in Safety-critical Systems

- **Assurance**: Making statements about properties of a system and demonstrate that we have some adequate level of confidence that statements are true.
- In safety-critical systems, we want to *assure* that the system is safe, secure and dependable
- Traditionally, assurance was implicit
  - Developers would provide assurance by providing the documentation and test results of the system
- With MDE, assurance can be explicit
  - We will talk about this throughout the course

Takeaways!

- A number of SDLC models are being used and have been transformed across the years
  - Our focus is safety critical systems, for which spiral and V model are used often
  - In particular, the spiral model emphasizes the iterative nature of hazard analysis
  - Of course, we can use other SDLC models, but keep hazard/risk analysis in mind
- MDE is prevalent in many industries, specially for safety-critical systems
- Explicit assurance is needed to ensure the systems we develop are safe, secure and dependable

McMaster University | Engineering

Visit our engineering grad studies fair!

Interested in exploring graduate studies after you graduate? Visit our upcoming fair to meet with staff, students, and faculty from across Canada!

September 18, 2025

1:00 p.m. - 4:00 p.m.

JHE Lobby

Scan to learn more

Brighter World

Next Week!

- Project:
  - You will get your board
  - Team contracts
  - Two more tutorials
- Requirements!