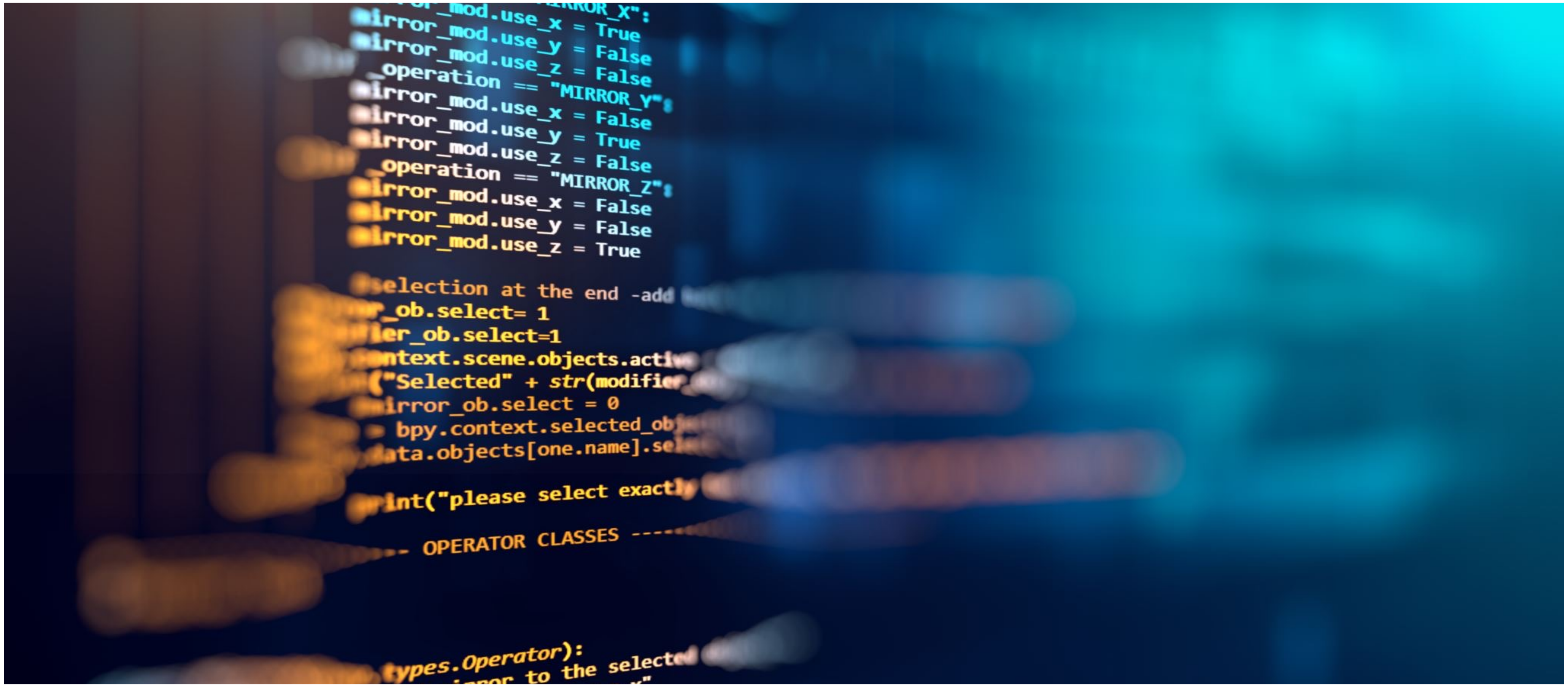


SOFTWARE DEVELOPMENT METHODOLOGIES

CM1301 – Week 5

Dr Carolina Fuentes Toro

FuentesToroC@Cardiff.ac.uk

Rules of engagement

- Email:
  FuentesToroC@cardiff.ac.uk
  Subject: CM1301 Query
- Office Address: Abacws Building, Room 3.52
- Office Hour : Wednesday 11:00-12:00 (Teams)
- Information on Learning central

Week 5, 6, 8, 11

- Software Development Methodologies
- Project Management
- Agile
- Scrum
- Kanban
- Git for teams
- Team theory

Activities week #5

- Lecture – Software development methodologies
- Watch videos on different software development methods

    Waterfall: https://www.youtube.com/watch?v=LxEmGNgqYJA

    Agile: https://www.youtube.com/watch?v=GzzkpAOxHXs

- Read the article: how Spotify has scaled agile to work in larger companies (**Link**)
- When to use agile versus waterfall

Software Development Methodologies

- Waterfall Model
- Prototype
- Iterative and Incremental development
- RUP
- Agile

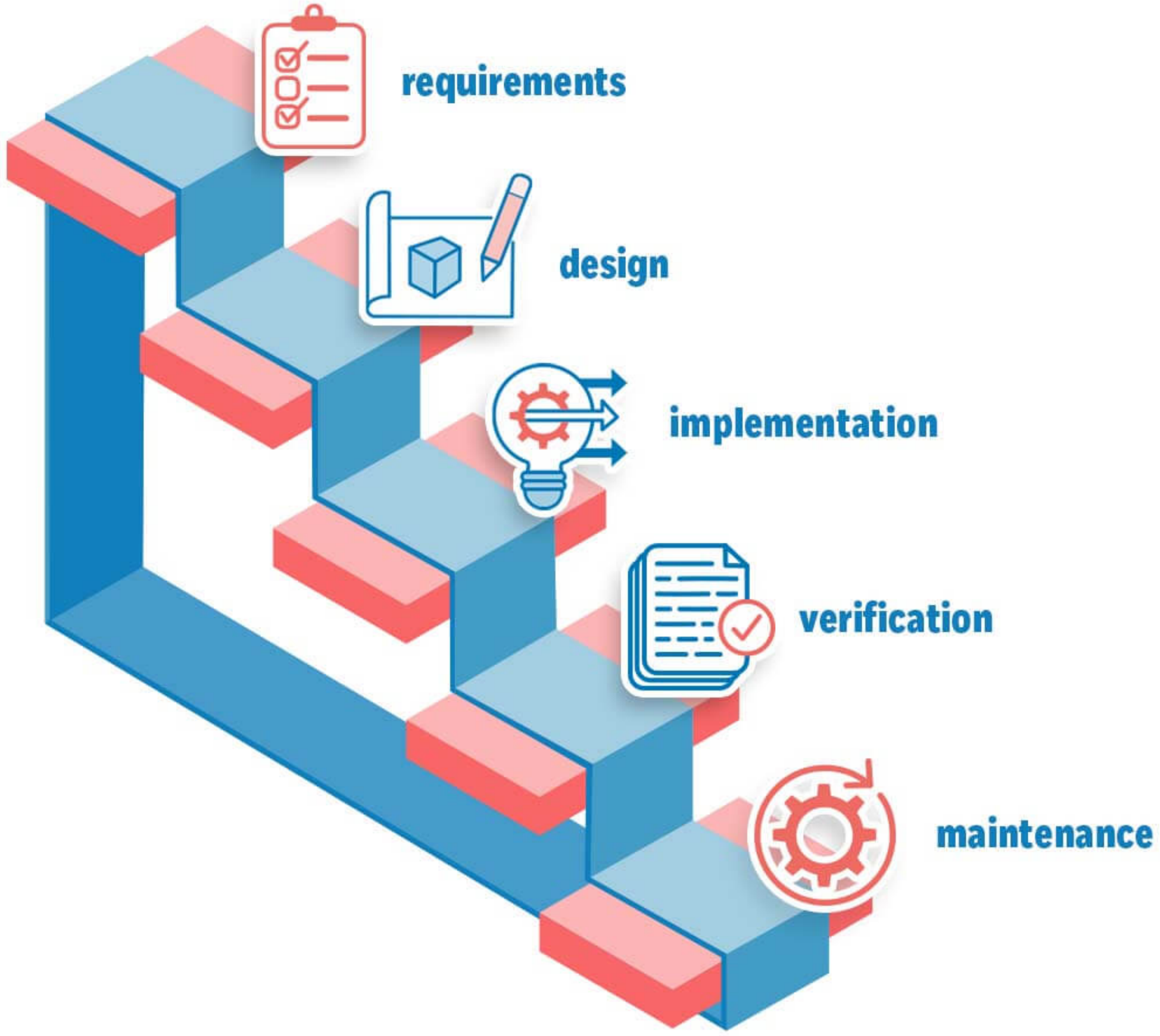
Software Development Methodologies

- What is the Software Process Model
  - Waterfall Model
  - Prototype
  - Iterative and Incremental development
  - RUP
  - Agile

Choosing a Model

- Consider your understanding of the requirements
  - Will collecting these be easy or challenging
- Expected lifetime of the project
  - Will the software require maintaining
- What is the level of risk?
- Scheduling constraints
- Interaction with management & customer
- Expertise of the development team

1. The Waterfall Model



Pros

- Works well when requirements are understood well and are stable
- Allows for easy testing and analysis
- Structured approach, easy to understand, and functional.

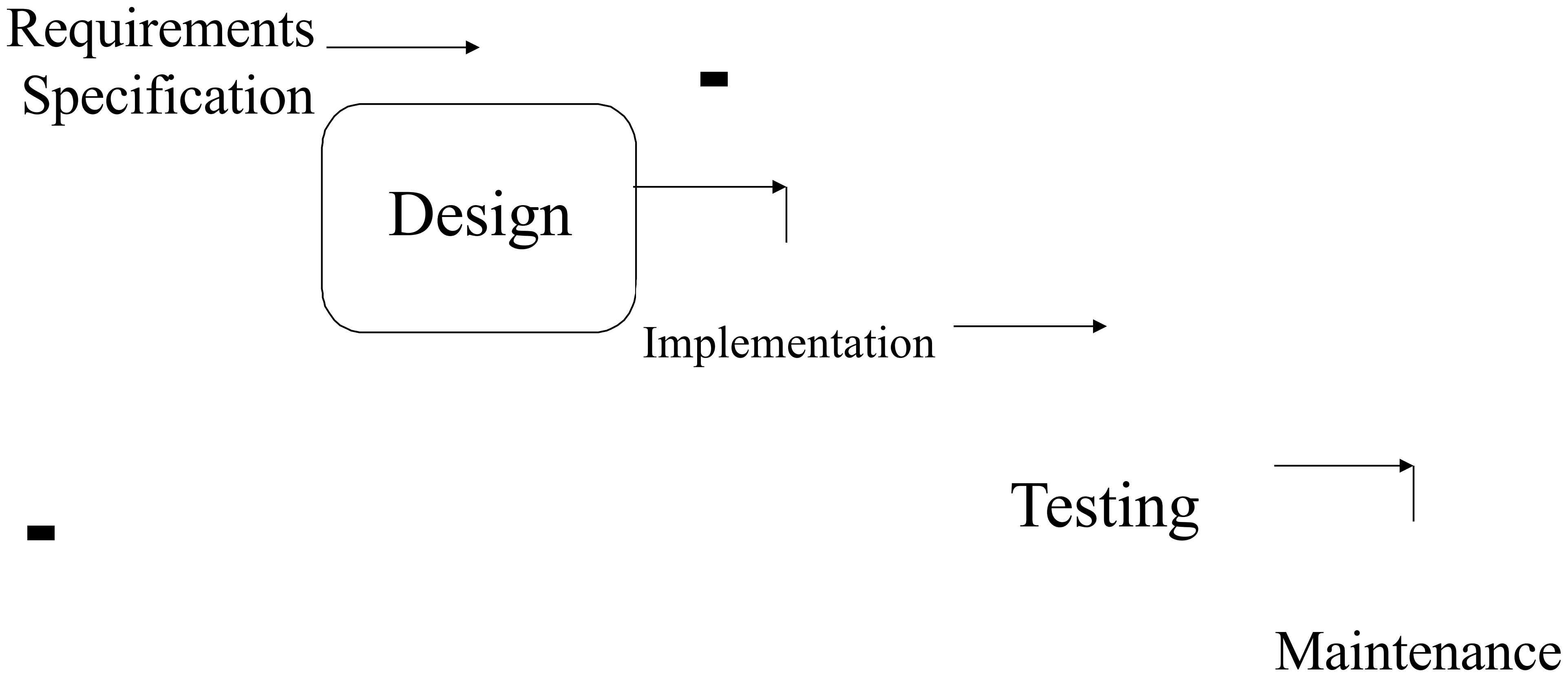
Cons

- Not very flexible
- Not suitable for project where programmes are not domain experts
- Not suitable for long and ongoing projects
- Only matches precise needs

1. Waterfall

The Waterfall Model

- &+ Documentation heavy



2.Prototyping

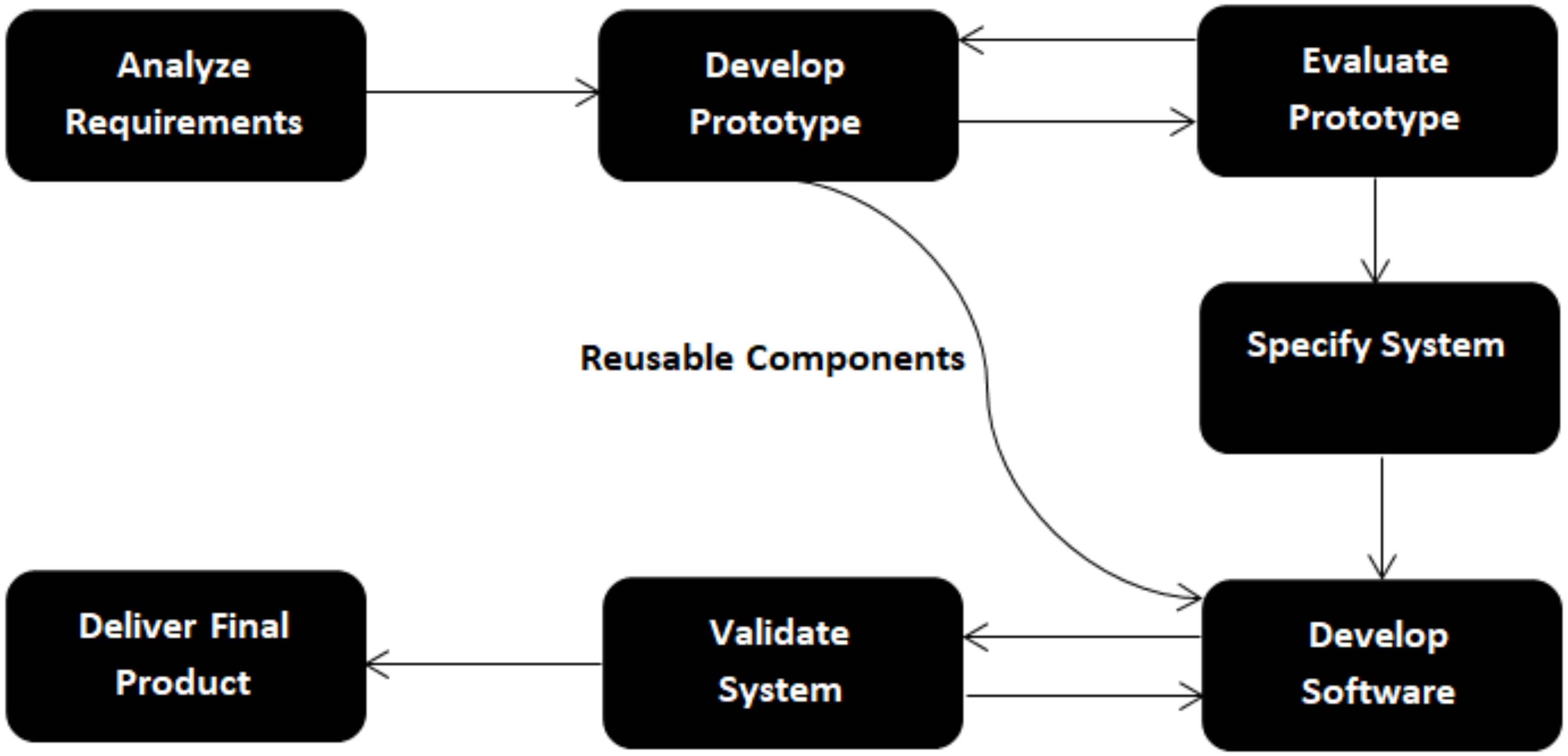
- Is based on the idea of inviting user feedback on an initial implementation and refining development on the basis of this.
- Aims to get past any difficulties with eliciting requirements
  - Identifies misunderstanding,
  - reveals missing facilities,
  - shows up difficult to use or confusing facilities
  - Incomplete, inconsistent requirements.
  - Creates acceptable user interface
- Demonstrate feasibility and usefulness.
- User training,
- To establish that some new technology will provide facilities.

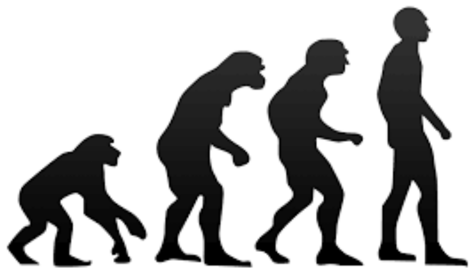
2.Prototyping

2.1 Throwaway

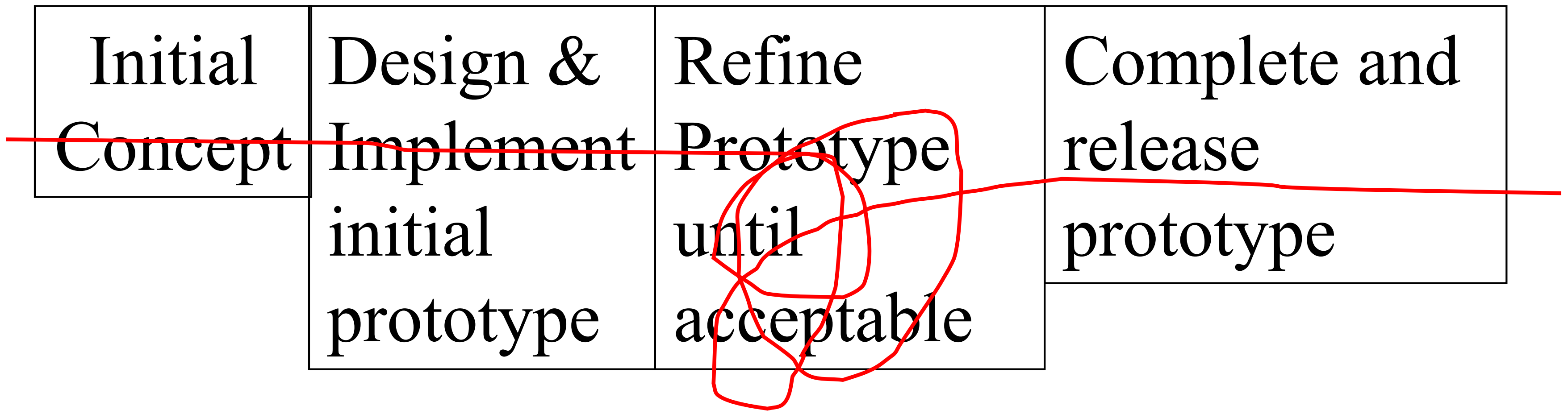
- Used to refine the requirements
- Particularly useful for systems with an emphasis on the user interface.
- Inviting user feedback
- Often used when new technology involved

Throwaway

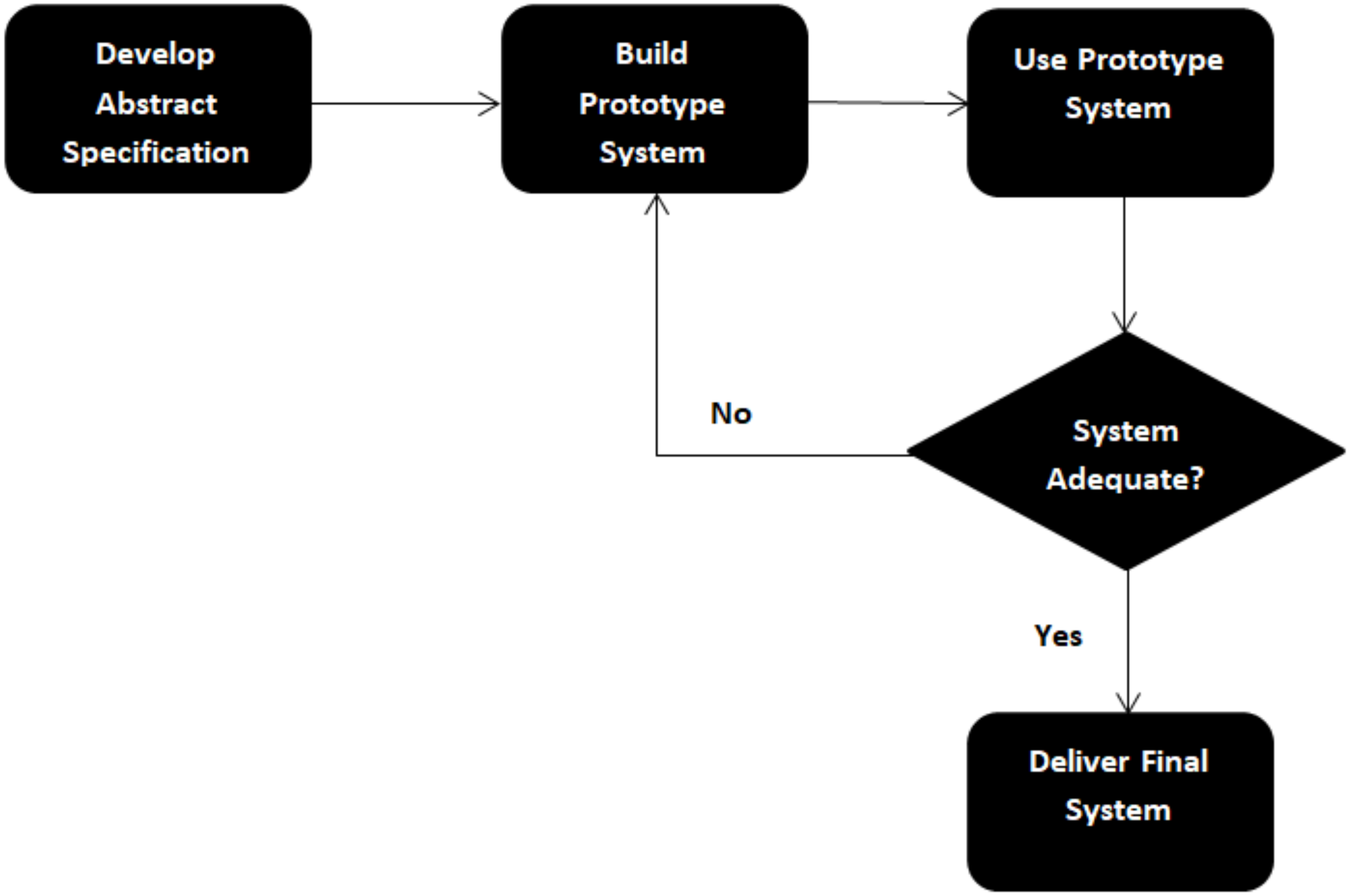

Analyze Requirements
Develop Prototype
Evaluate Prototype
Specify System
Reusable Components
Develop Software
Validate System
Deliver Final Product



2.2 Evolutionary Prototyping


Initial Concept
Design & Implement initial prototype
Refine Prototype until acceptable
Complete and release prototype

- Start by developing the parts they understand
- Develop initial implementation exposing it to user comments / feedback.
- Refine it through repeated stages until an adequate system has been developed.


Develop Abstract Specification
Build Prototype System
Use Prototype System
System Adequate?
No
Yes
Deliver Final System

Pros

- Gives clear idea about the functionality of the software
- Reduces risk of failure in a software functionality
- Assists well in requirement gathering and overall analysis.

Cons

- Difficult to plan how long the project will take to complete.
- Excessive client involvement can affect the processing.
- Too many changes can affect the workflow of the software

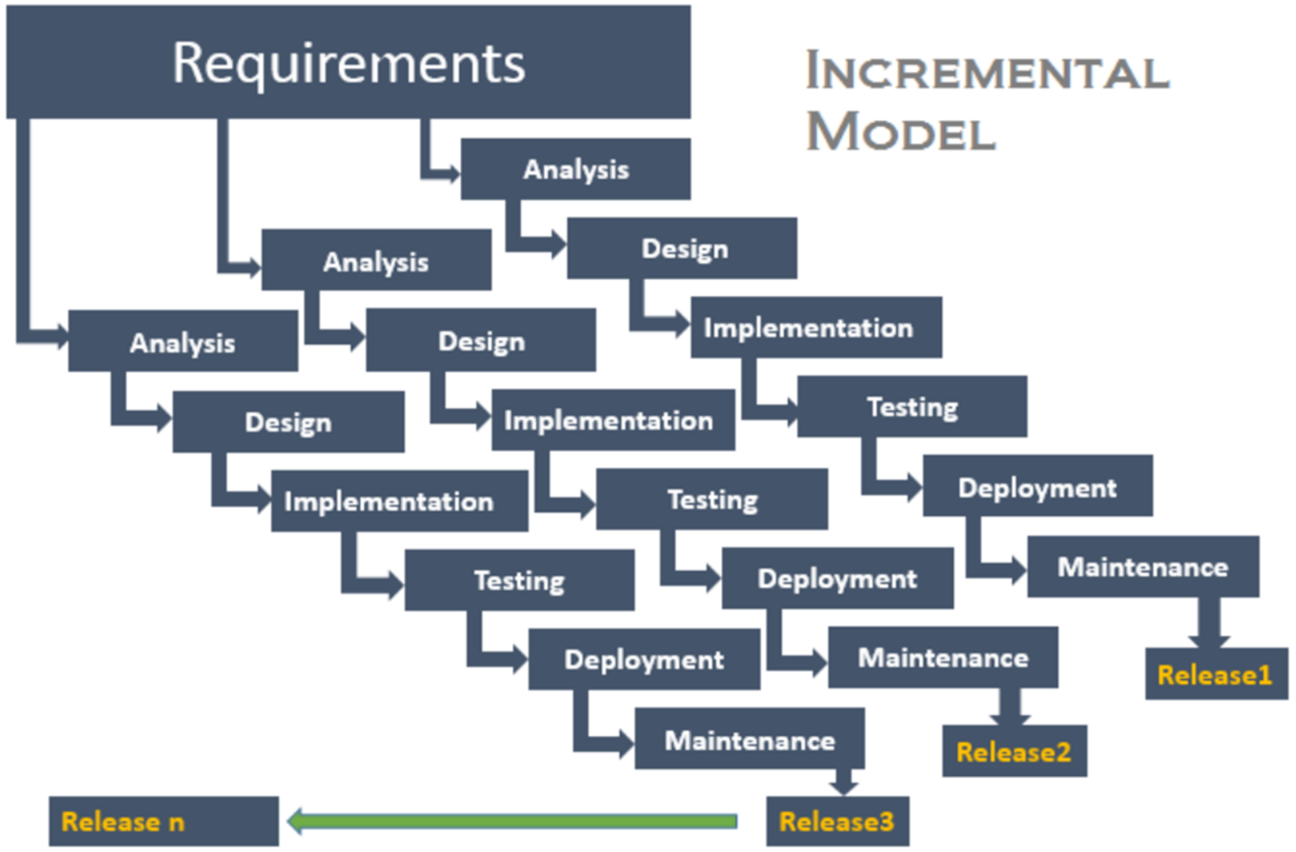
3.1 Iterative development

- Breaking down the software development of a large application into smaller chunks.
- The purpose of working iteratively is to allow more flexibility for changes.
- feature code is designed, developed and tested in repeated cycles.
- With each iteration, additional features can be designed, developed and tested until there is a fully functional software application ready to be deployed to customers.
- Typically iterative development is used in conjunction with incremental development.

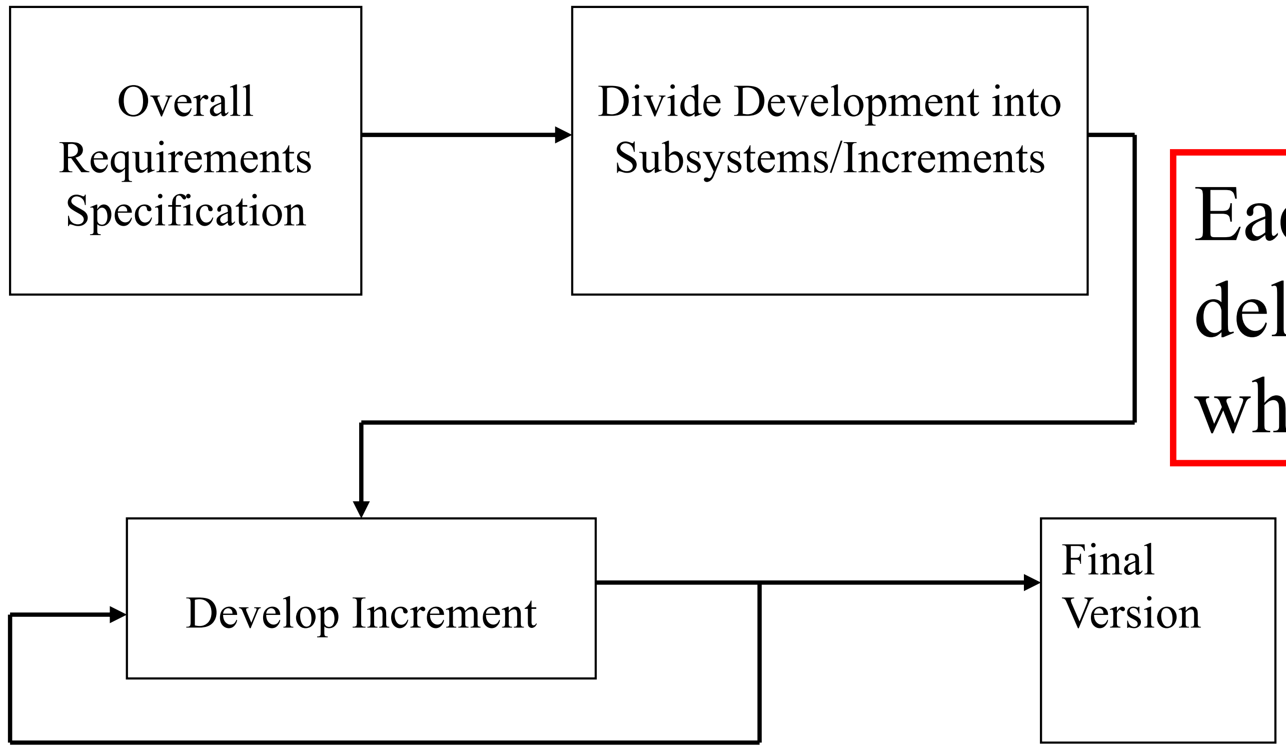
3.2 Incremental Development

Where used

- Used for the development of large systems where requirements may be subject to change.



Incremental Development


Overall Requirements Specification
Divide Development into Subsystems/Increments
Develop Increment
Final Version

Each increment delivers a product which can be used.

After each increment users can evaluate the system and provide feedback.

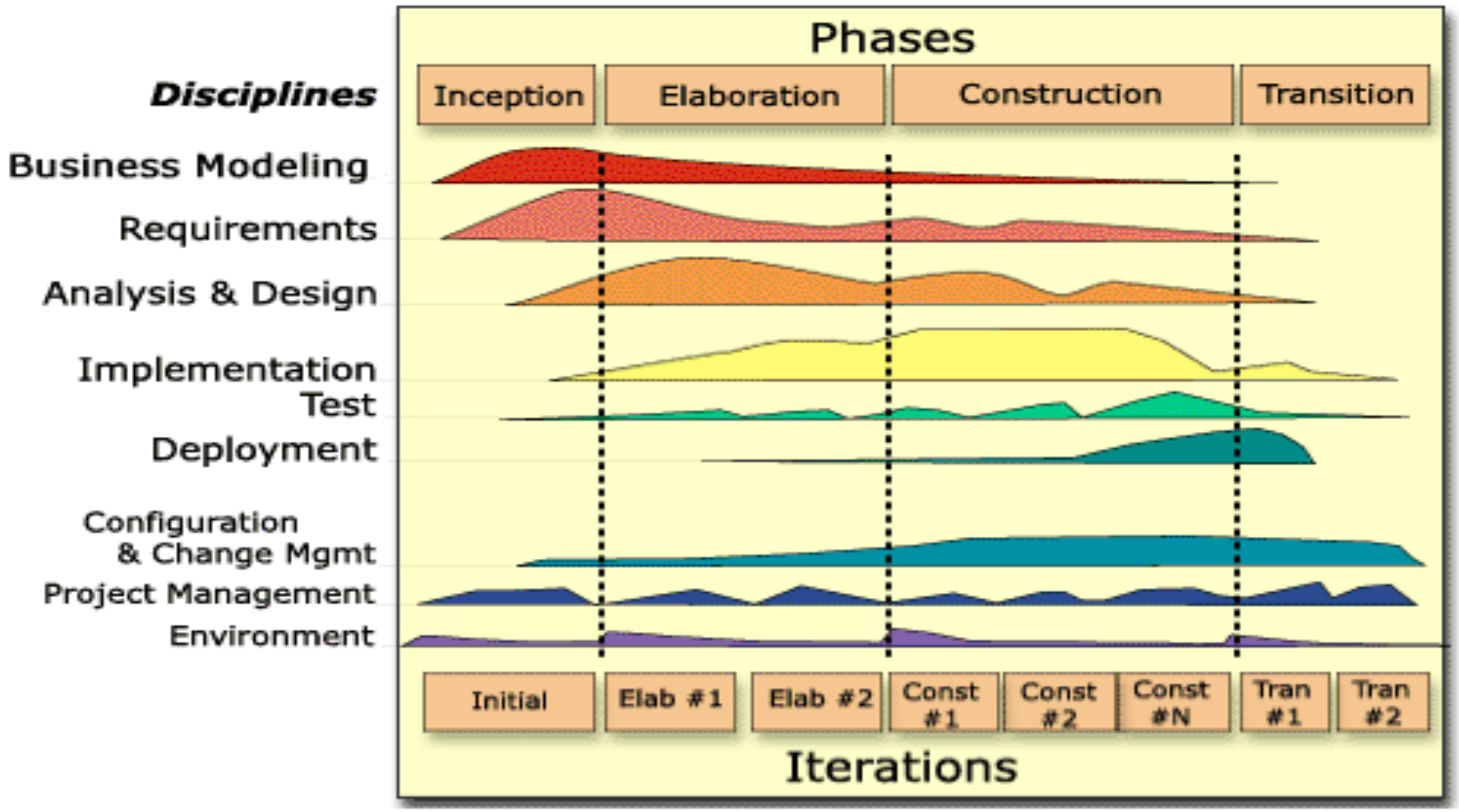
Pros

- Core capabilities are delivered early in the project;
- Core capabilities can be evaluated by the customers early in the project;
- A ‘safe’ approach

Cons

- Can be difficult to split the problem into appropriate increments;
- System architecture has to be established before requirements are complete
- Extra time must be spent on testing, documenting and maintaining ‘temporary’ products until the full system is delivered.

4. The Rational Unified Process (RUP)



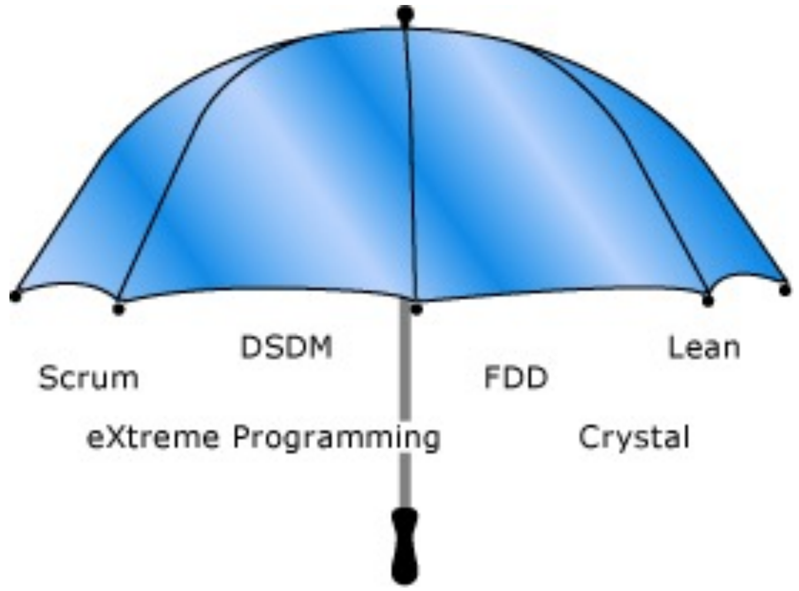
- A risk-driven, UML use-case-based, iterative development process

5. Agile development methodologies

- Iterative and incremental development are key practices in Agile development methodologies.
- Agile methodologies, the shorter development cycle, referred to as an <u>iteration</u> or <u>sprint</u>, is <u>time-boxed</u> (limited to a certain increment of time, such as two weeks).
- At the end of the iteration, working code is expected that can be demonstrated for a customer.

Agile development is not a methodology in itself. It is an umbrella term that describes several agile methodologies.

Initially at the signing of Agile Manifesto in 2001, these methodologies included Scrum, XP, Crystal, FDD, and DSDM. Since then, Kanban has also emerged as a valuable agile methodology .



Agile Principles and Values, by Jeff Sutherland

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

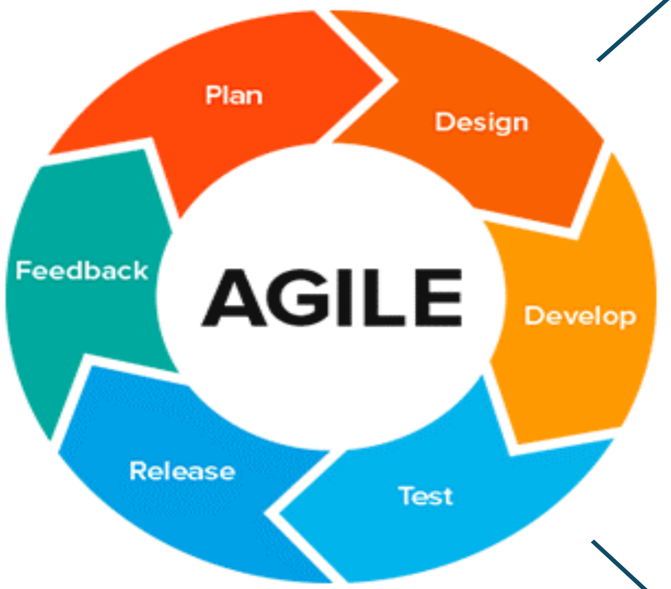
Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

http://www.agilemanifesto.org/



XP

- Lightweight
- Discipline
- Humanistic focus
- It has values and principles
  - Communication, pair programming, simplicity, feedback

SCRUM

- Specific methodology for PM
- Agile methodology used by project managers
- team assess progress in time-boxed daily meetings of 15 minutes
- At the end of the sprint, the team holds two further meetings: the sprint review which demonstrates the work done to stakeholders to elicit feedback

KANBAN

- aims to manage work by balancing demands with available capacity
- 6 rules
  - Never Pass Defective Products. ...
  - Take Only What's Needed. ...
  - Produce the Exact Quantity Required. ...
  - Level the Production. ...
  - Fine-tune the Production or Process Optimization. ...
  - Stabilize and Rationalize the Process.

Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Principles

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Philosophy - a little at a time

- The plan, design and team changes a little at a time
- Don’t design, plan or code more than is needed at the moment so that options for the future remain open.
- Do the simplest thing that works and meets current needs, don’t build in extra complexity ‘in case’. **MVP – Minimum Viable product**
- As each ‘piece’ is added developers learn what works and what doesn’t.
- Customer learns what value the system offers and what features are needed next.
- The team should take pride in delivering high quality.

XP – Extreme Programming

- Extreme programming is identified by the fact that customer involvement in the software development process is unbelievably high.
- Coding
  - Pair Programming (Navigator and observer)
  - Code review - https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/
- Test Driven development
  - using Automated unit testing
  - Acceptance testing
  - System wide integration testing

Next lecture Week #6

- Scrum and Kanban