CLEMSON UNIVERSITY

CPSC 3720 Lesson 2:

The Tar Pit, SDLC and Agile Intro

Connie Taylor
Professor of Practice

School of COMPUTING

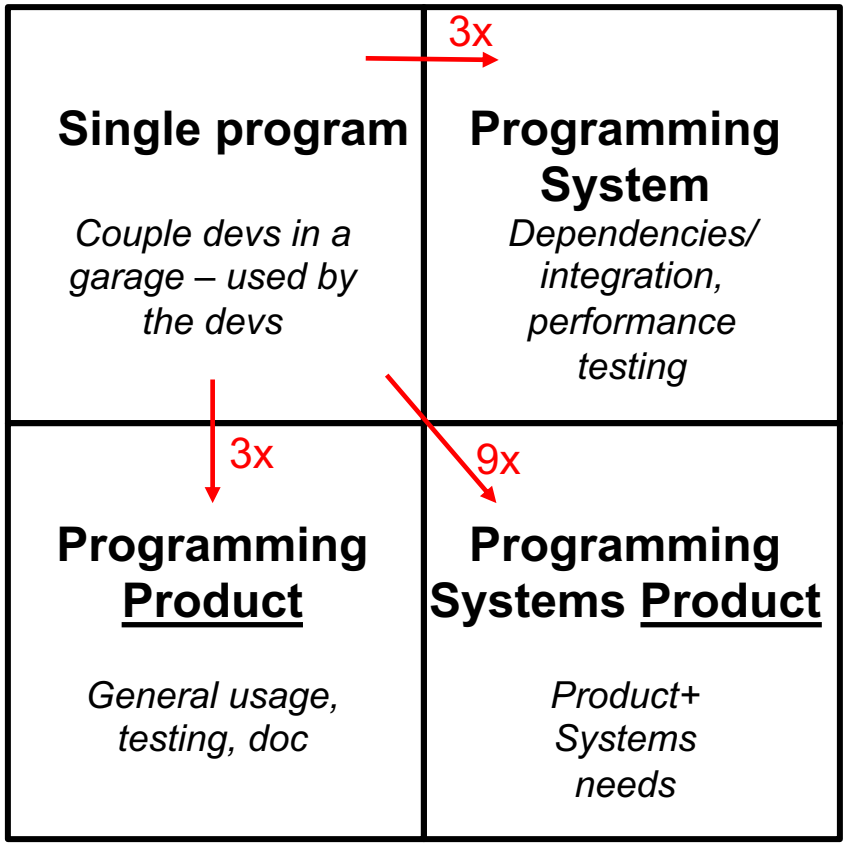
Today's Objectives

- Review reading/discussion
- Discuss software engineering and complexity
- Begin discussion of software lifecycle and processes

Homework Discussion Breakouts

- At your tables discuss the following questions regarding your homework reading. Be ready to report back afterwards (~5 min)
  - **Discussion Question 1:** Can you think of a "world/industry" that will not be eaten by software? Justify your answer.
  - **Discussion Question 2:** Do you agree with Fred Brooks' perspective that software programming is a craft? Or do you think it is more of an engineering discipline? Why?

The Tar Pit – Complexity of a Program vs. Product



Software Development Process

Software Process: a way of breaking down this overall software development work into manageable sub-tasks; systematic and somewhat formal

Building a House

It is generally accepted that it is easier to estimate and build a quality house than it is to estimate and build a quality complex software system –**why**?

The Tar Pit – Complexity of a Program vs. Product

Single program *Couple devs in a garage – used by the devs*	**Programming System** *Dependencies/ integration, performance testing*
Programming <u>Product</u> *General usage, testing, doc*	**Programming Systems <u>Product</u>** *Product+ Systems needs*

3x

3x

9x

The Tar Pit – Complexity of a Program vs. Product

Single program *Couple devs in a garage – used by the devs*	**Programming System** *Dependencies/ integration, performance testing*
Programming <u>Product</u> *General usage, testing, doc*	**Programming Systems <u>Product</u>** *Product+ Systems needs*

3x

3x

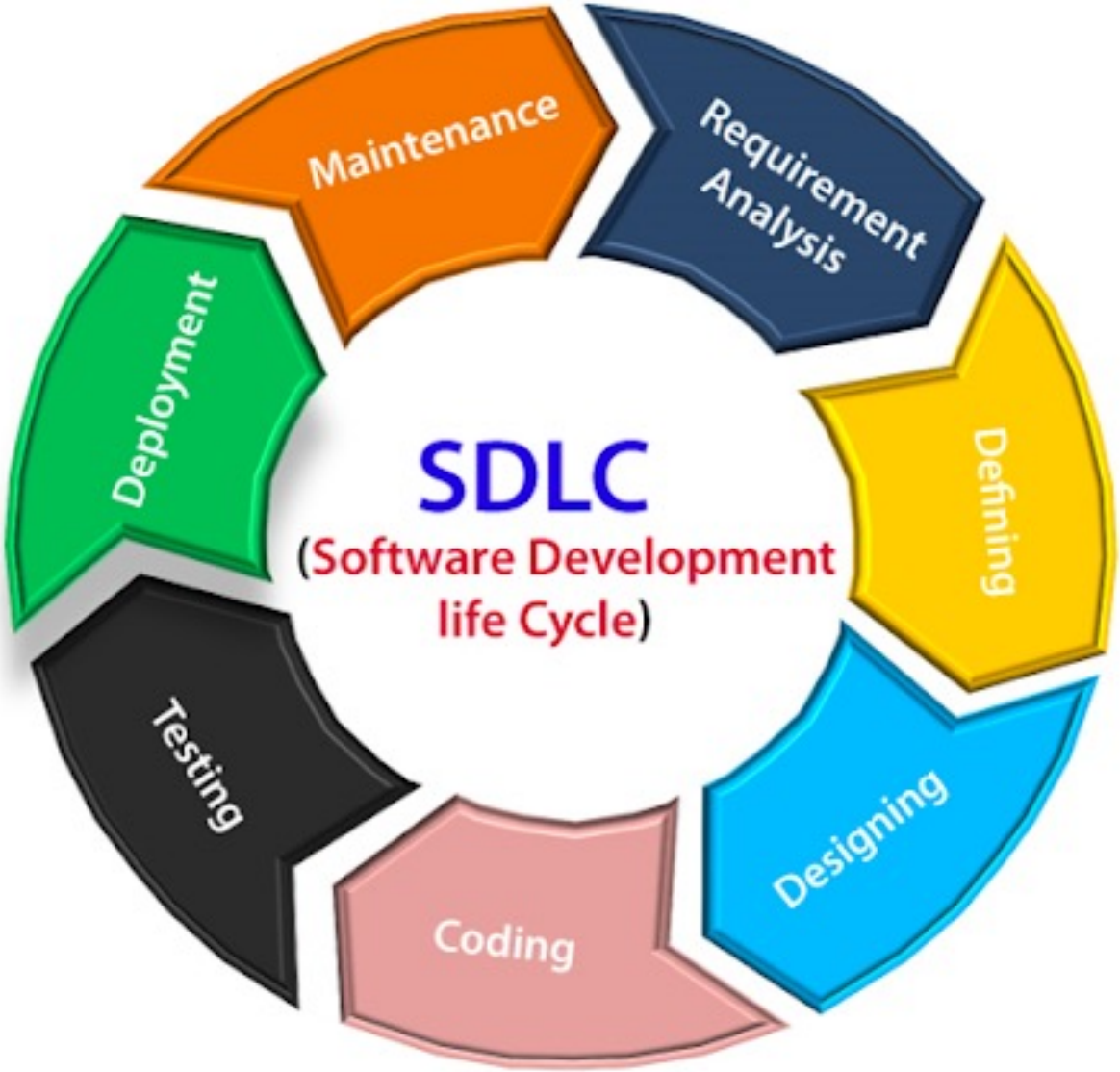
9x

How do we manage this complexity??

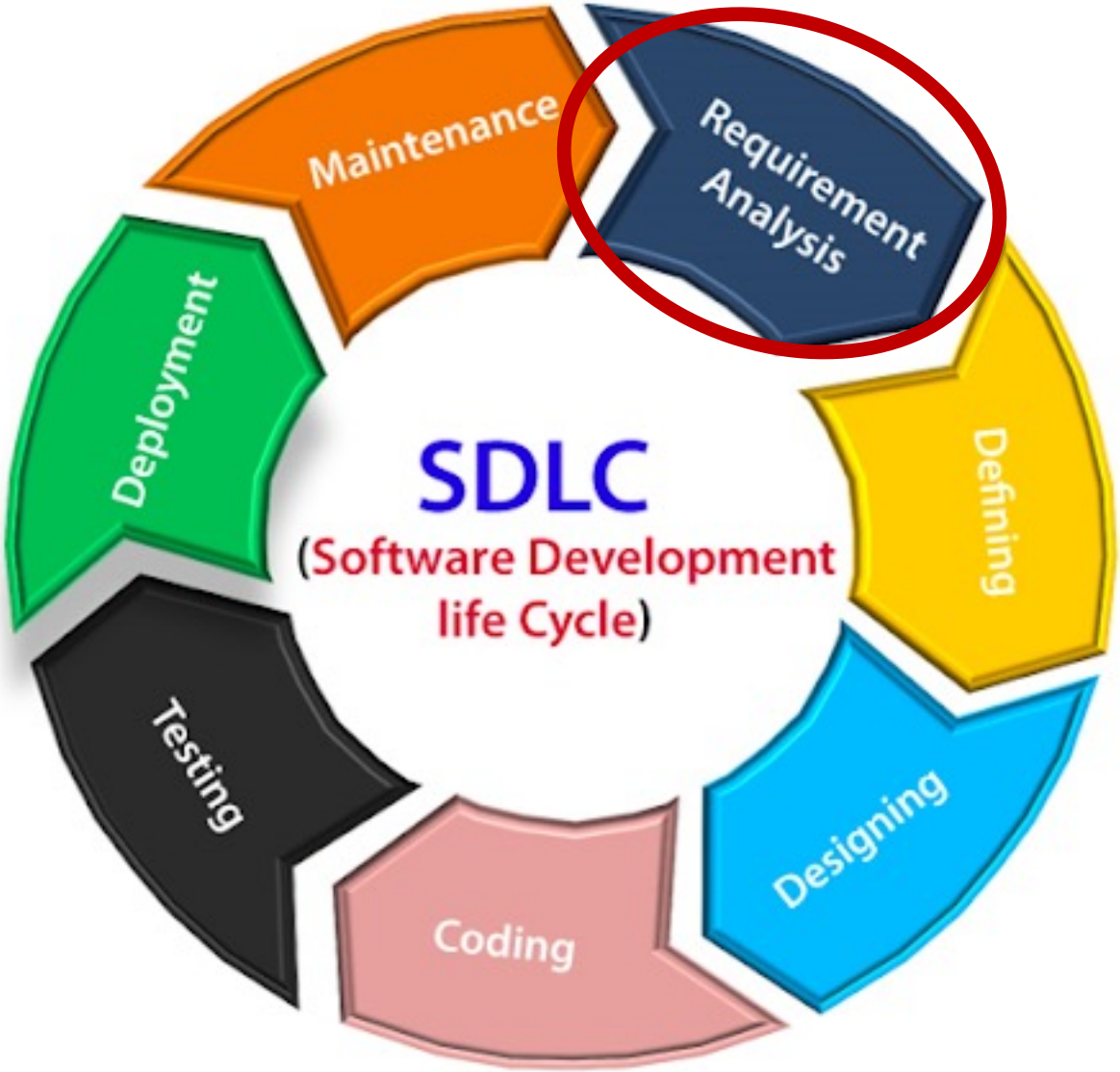
Software Development Process

Software Process: a way of breaking down the overall software development work into manageable sub-tasks; systematic and somewhat formal

Software Development Process Steps

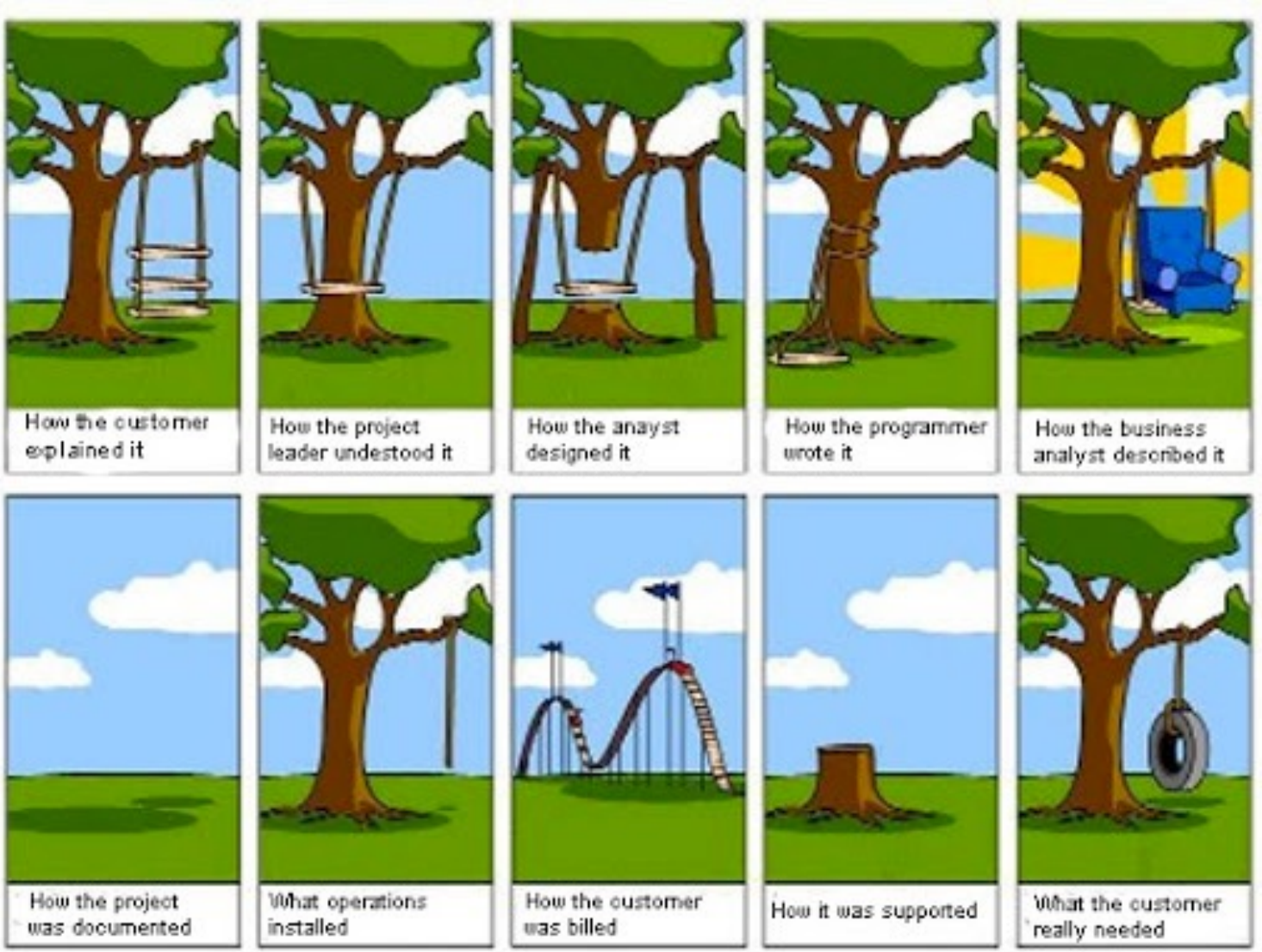


Requirements Analysis



SDLC: Requirements Analysis

Requirements Analysis



SDLC: Requirements Analysis

- The WHAT? and the WHY?
- Understanding what the customer wants or what they "think" they want (Ask WHY?)
- Focus on the business problem you are trying to solve
- Understand what is most important to the customer to enable prioritization
- Can be documented in various ways depending on the process:
  - Formal requirements specifications
  - Wireframes
  - Use case documents
  - Prototypes

So How Do We Know We Got it Right?

How do we validate the requirements?

Avoid producing a good apple when an orange is required

Validation vs. Verification

- Project Management Book of Knowledge (IEEE standard) defines these terms:

- **Validation**: The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers.

- **Verification**: The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process.

Validation vs. Verification

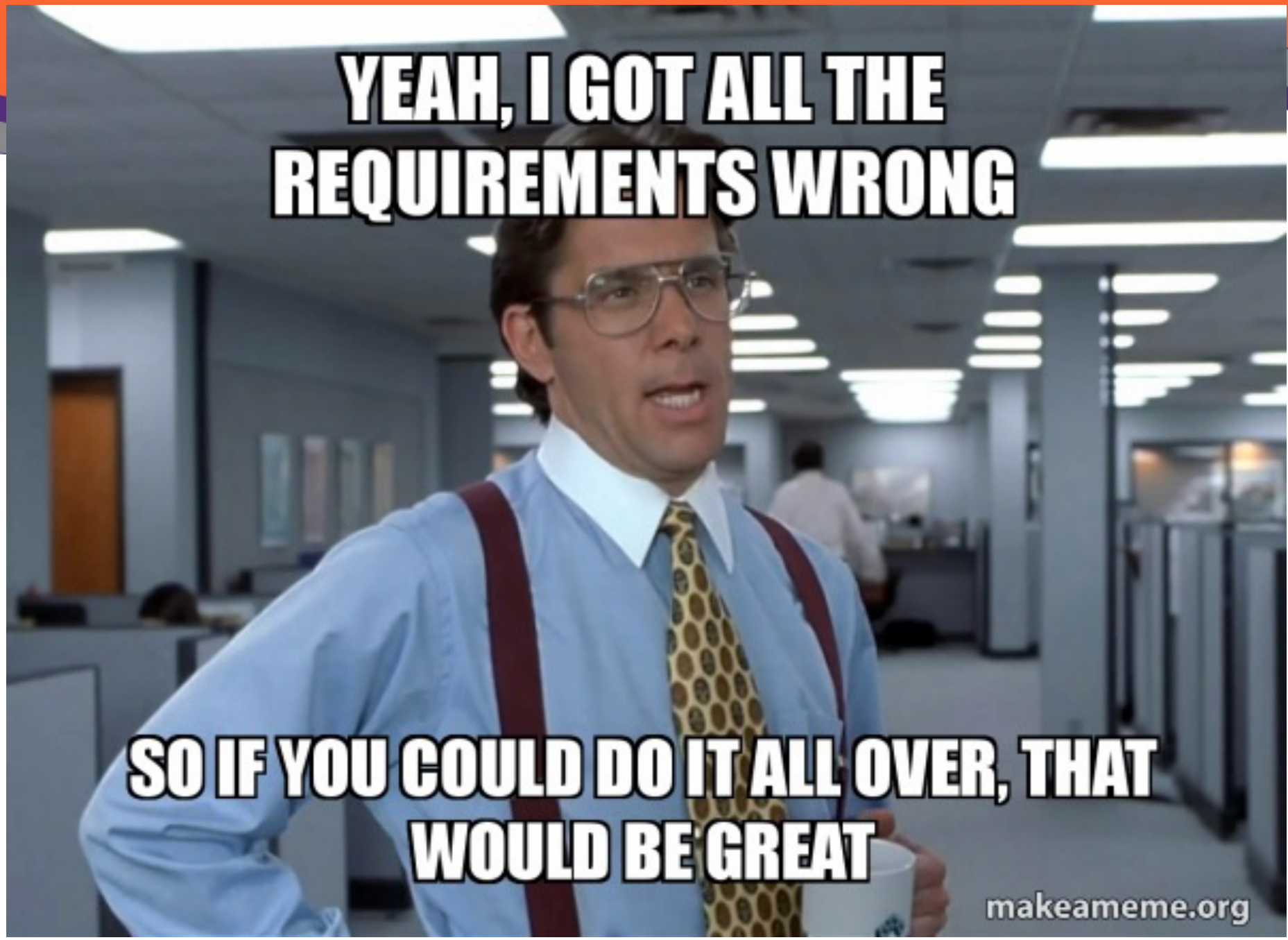
- **Validation**: Are we producing the Right product?

- **Verification**: Are we producing the product Right?

Discussion:

Which is more important?

Validation (producing the Right product) –or-
Verification (producing the product Right)

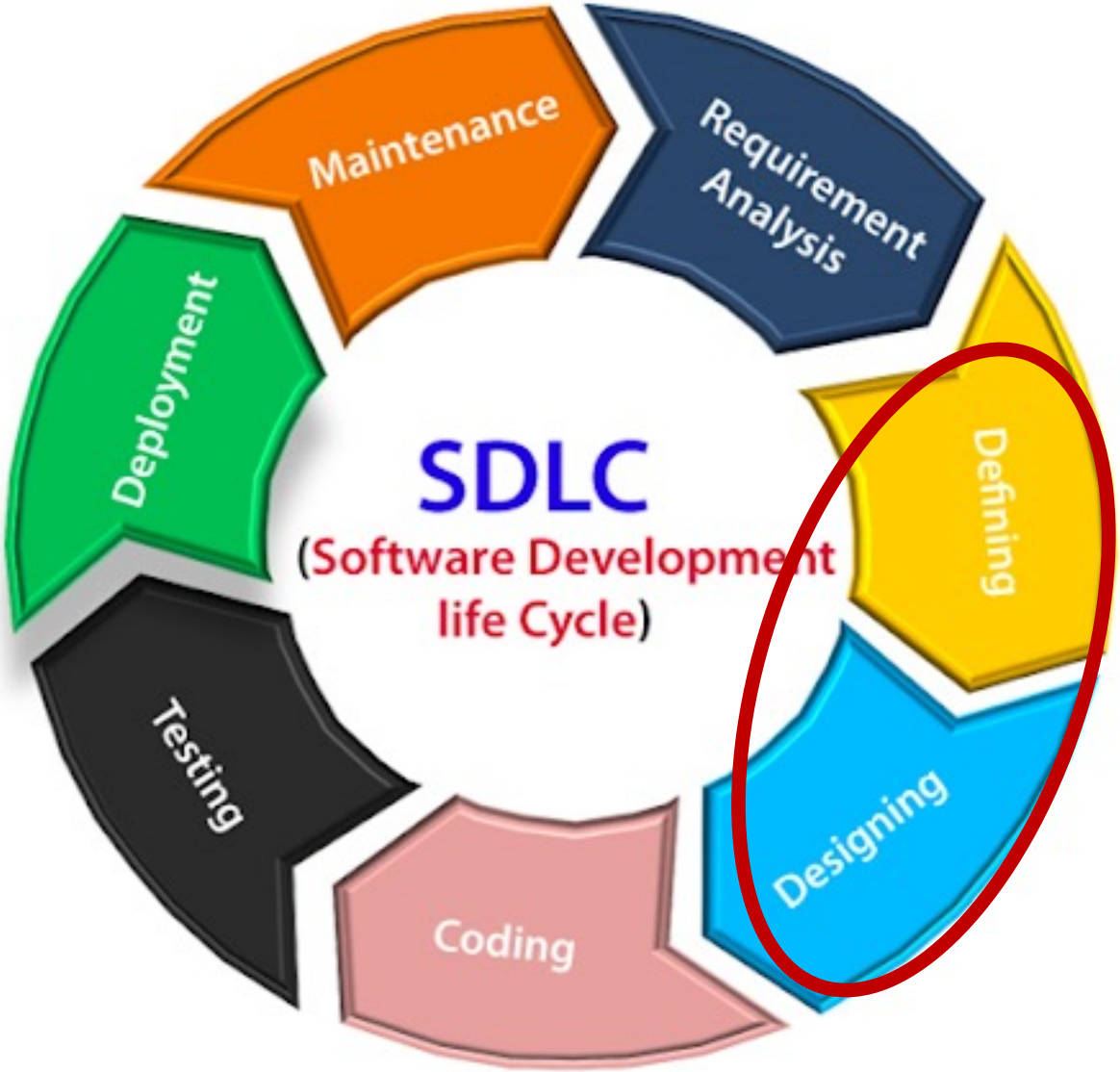

YEAH, I GOT ALL THE REQUIREMENTS WRONG
SO IF YOU COULD DO IT ALL OVER, THAT WOULD BE GREAT
makeameme.org

SDLC: Requirements Analysis

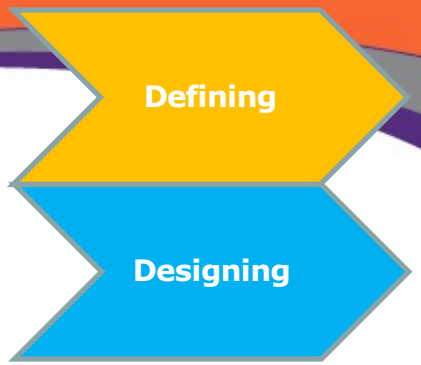
Prof Taylor's Rule of Software Engineering:

"Software engineering IS requirements discovery"

Define and Design

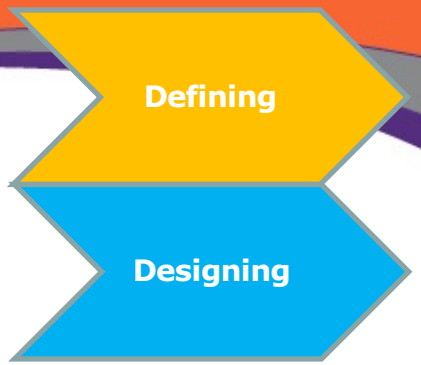


Define and Design



- The **HOW**?
- Need to understand both the business and non-functional requirements
- Depending on the type and size of system you will have various layers of design:
  - System architecture
  - Deployment architecture
  - Object/Class Designs
  - Database designs
  - UI designs

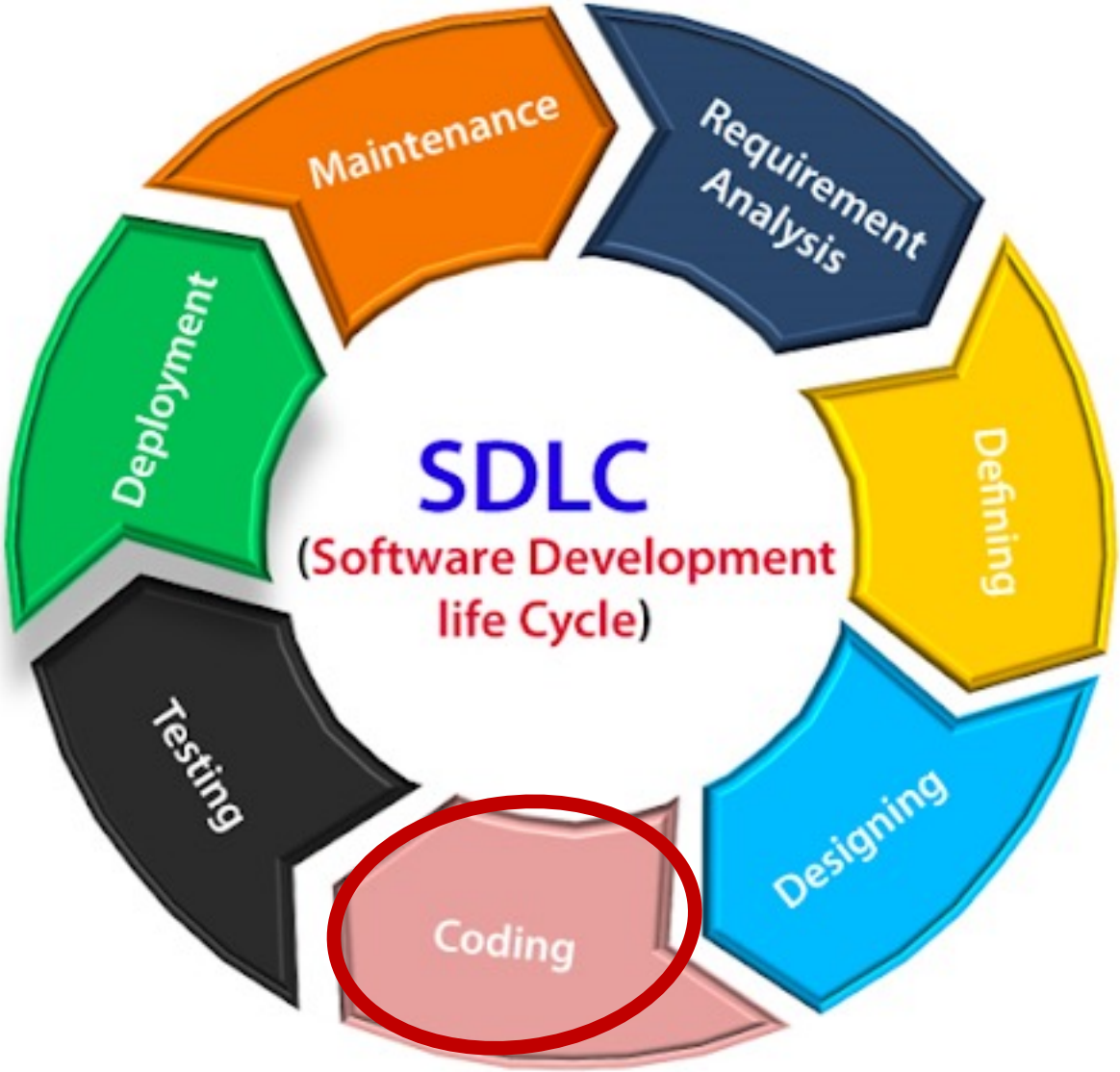
Define and Design



- Can be documented in various ways depending on the process and the type of system:
  - Formal design specifications
  - UML
  - State and Transition diagrams
  - **Wiki documents**
  - **Contract documentation (for APIs)**
  - ERDs (Entity Relationship Diagrams)
  - **Data Flow Diagrams (DFDs)**
  - **Whiteboarding sessions and photos!**

BOLD= what we will do in this class

Coding


Maintenance
Requirement Analysis
Deployment
Defining
SDLC
(Software Development life Cycle)
Testing
Designing
Coding

Coding

- AKA Implementation

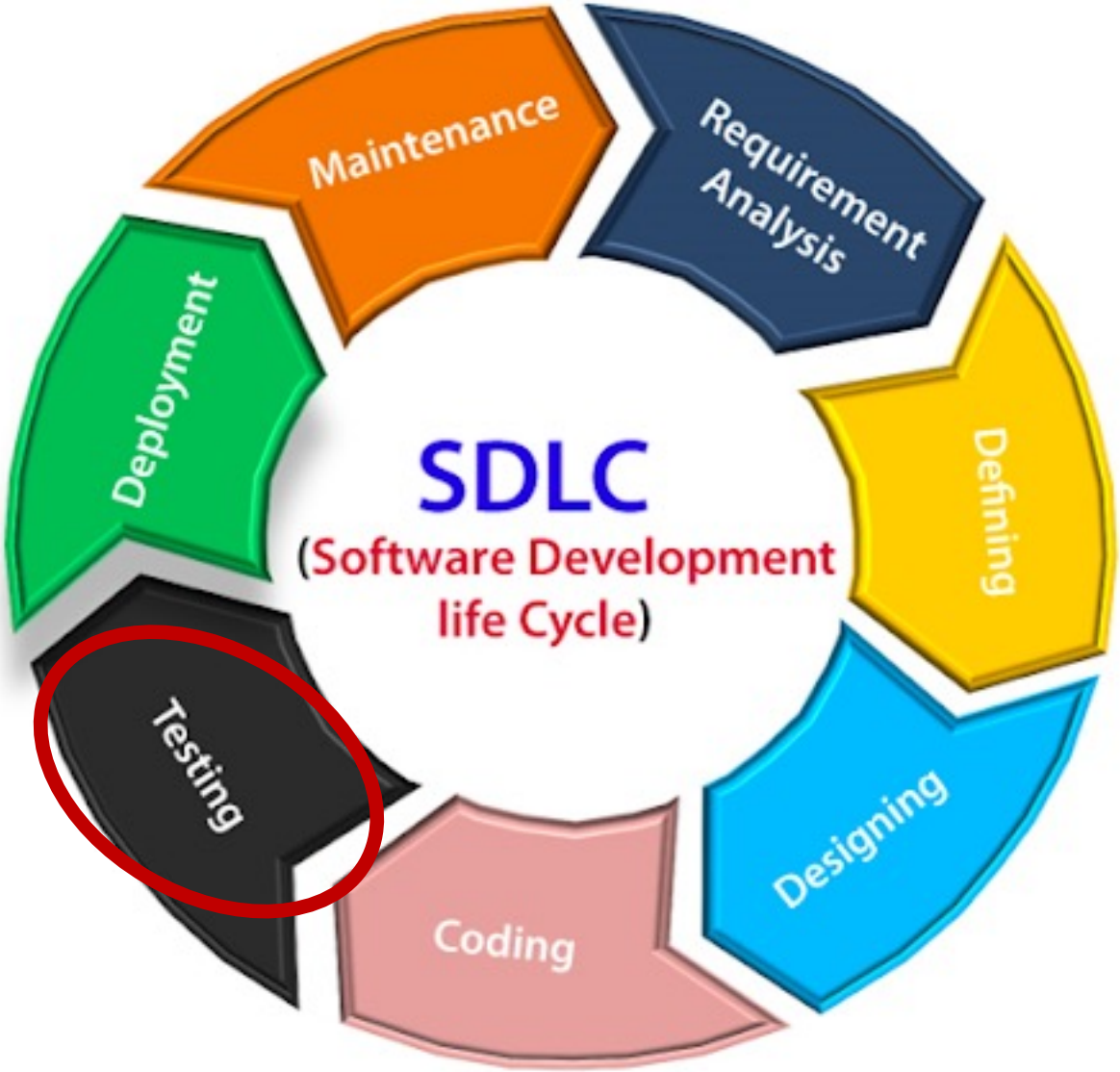
- Depending on organization you could have different standards and methods:
  - Language requirements
  - Pair programming
  - Code standards
  - Code reviews
  - Tool usage requirements
  - Internal and external frameworks
  - Unit testing requirements

Coding

- Dev/Coding Tools:
  - **Configuration management (GIT)**
  - IDEs
  - 3rd party tools – could be open or not (Log4J)
  - **API development tools**
  - **Documentation tools**

BOLD= what we will do in this class

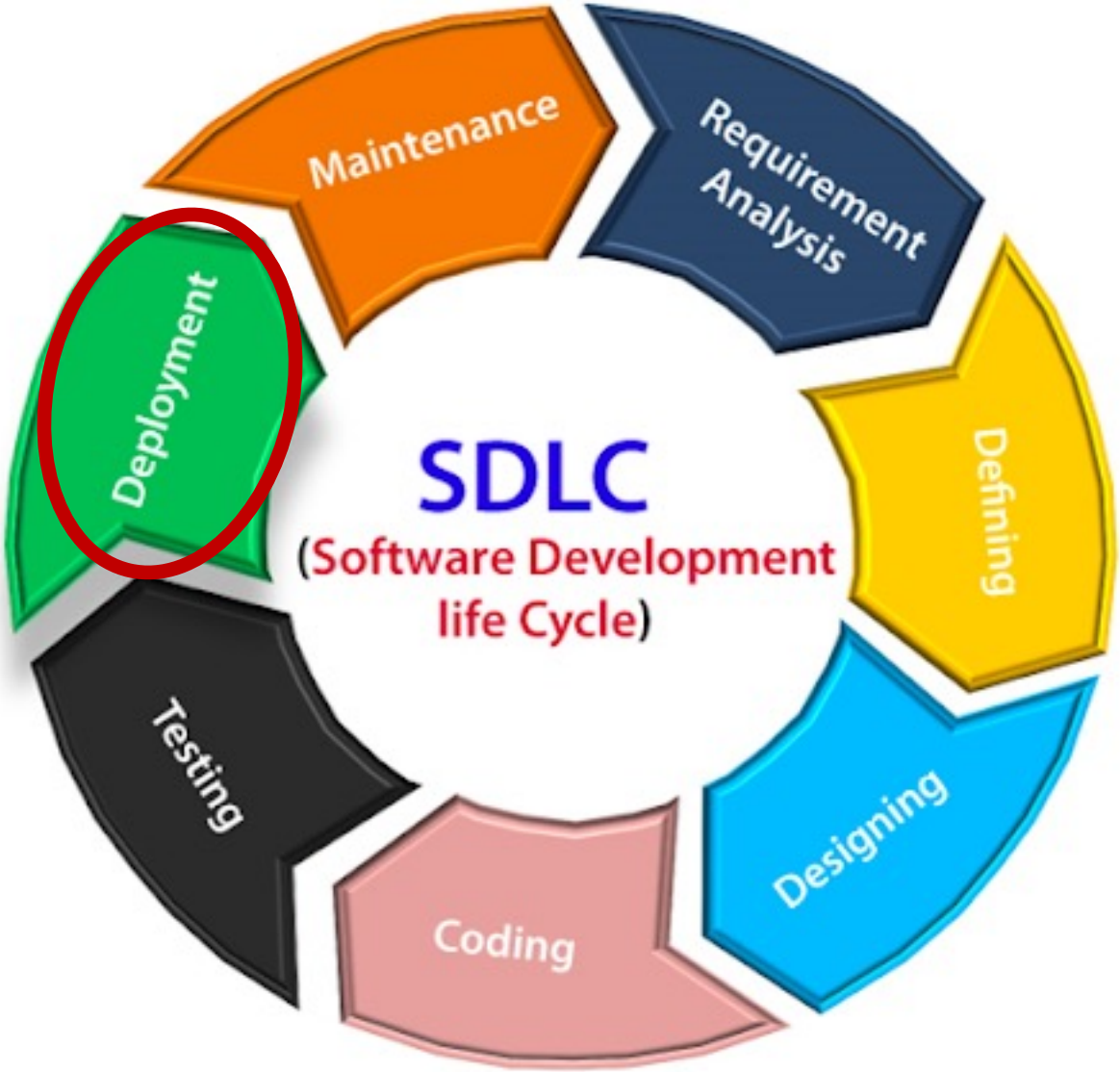
Testing


Maintenance
Requirement Analysis
Deployment
Defining
SDLC
(Software Development life Cycle)
Testing
Designing
Coding

Testing

- AKA Quality Assurance – the validation AND verification
- Depending on organization and process you will have different approaches:
  - When you test:
    - Unit Testing
    - Integration Testing
    - System Testing
    - Performance Testing
    - Regression Testing
  - How you Test
    - Test Driven Development
    - Continuous Testing
    - Automated vs. Manual

Deployment

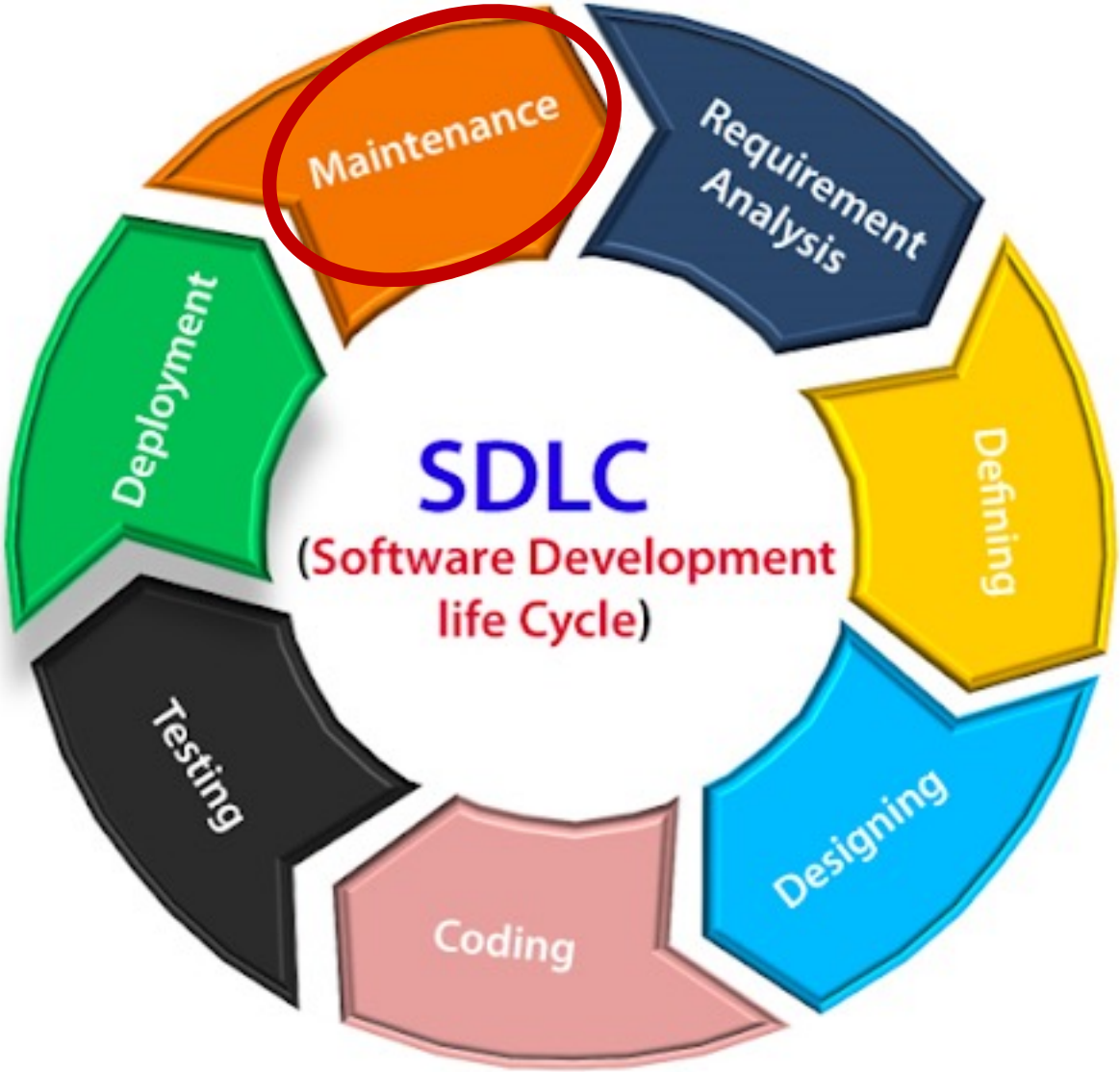

Maintenance
Requirement Analysis
Deployment
Defining
SDLC
(Software Development life Cycle)
Testing
Designing
Coding

Deployment

- Deployment is the process of putting your software into production for use by the customer

- Deployment will vary based on type of software being developed:
  - **Commercial "on-premise"** – Customer IT will deploy the software
  - **Commercial SaaS** – Company dev/operations will deploy the software
  - **Internal** – Company dev/operations will deploy the software

Maintenance



Maintenance

- Fixing bugs in production that are found by the customer
- The deployment model will dictate how the maintenance is delivered
    - Emergency Patch fixes
    - Maintenance Releases
    - Major Releases

-OR-

    - Continual Deployment (more about this later)

Putting it all Together

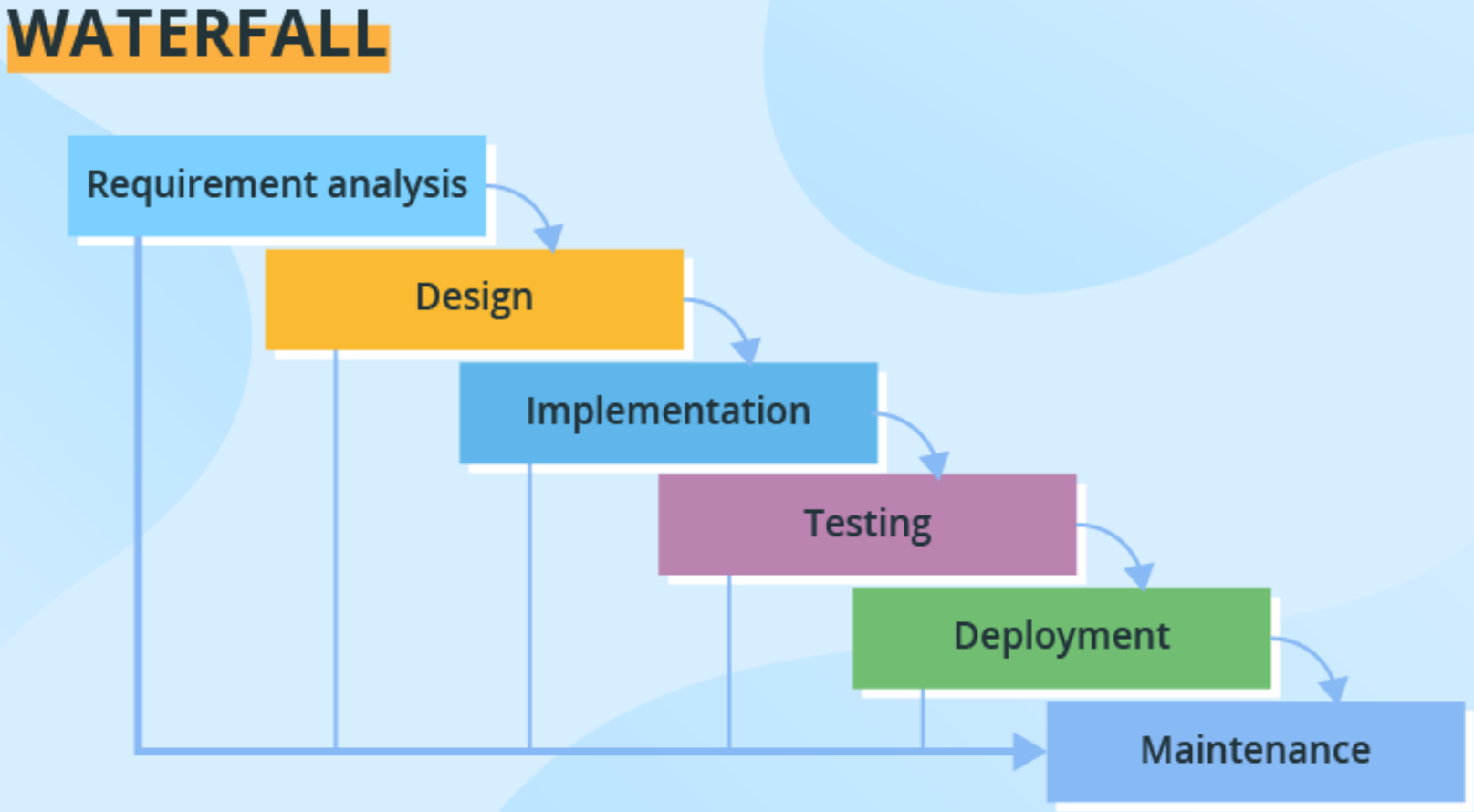
- The steps I just described are usually used in most software engineering projects; however, they can be done in many different ways
- The SDLC in use will vary company to company based on their business needs and culture; not a “one size fits all”

Process Models

- Waterfall
- V-Model
- Incremental
- Iterative
- Spiral Model
- RUP
- Agile:
  - Scrum
  - XP
  - Kanban
  - Lean

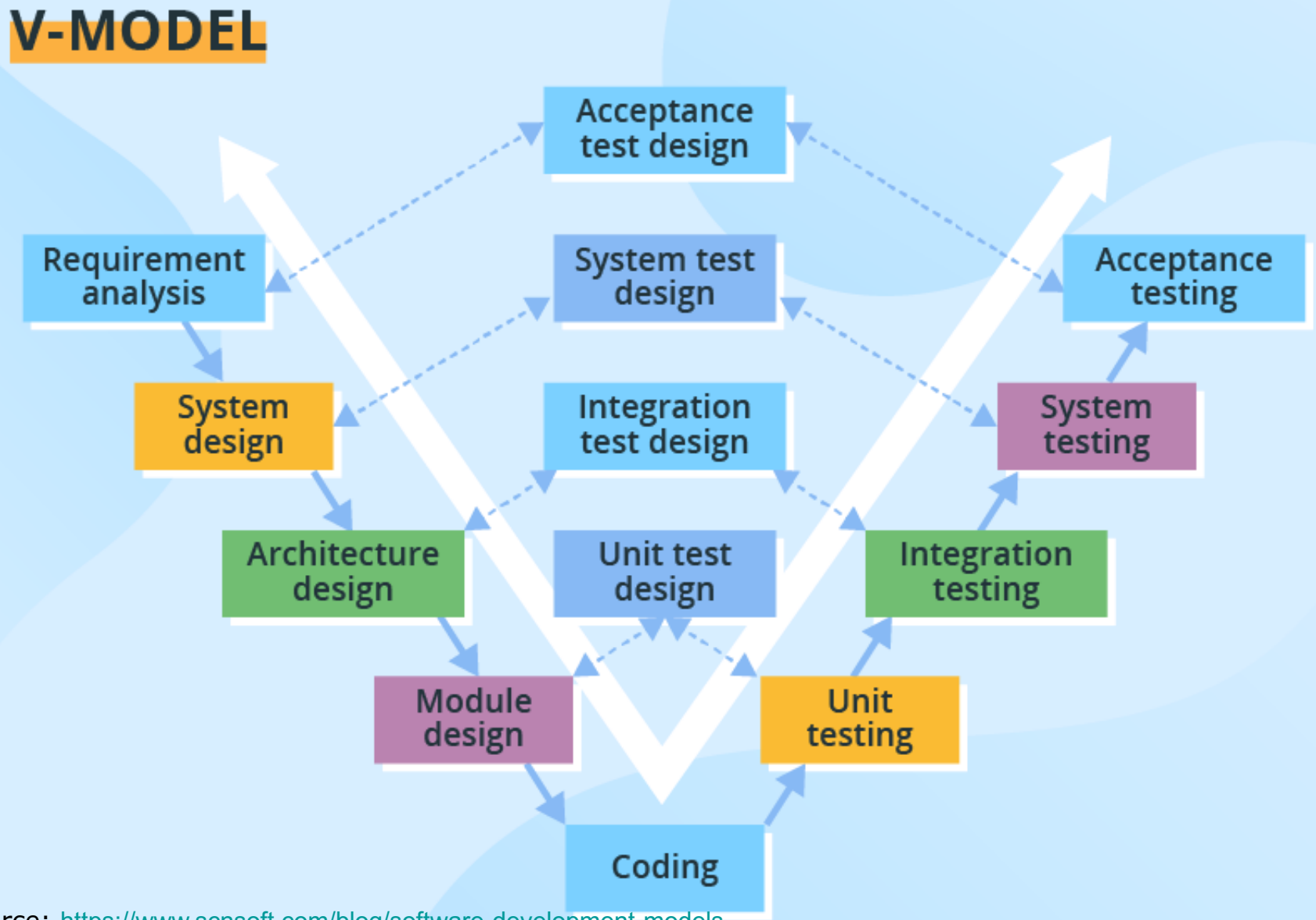
Source: https://www.scnsoft.com/blog/software-development-models

Waterfall Process Model



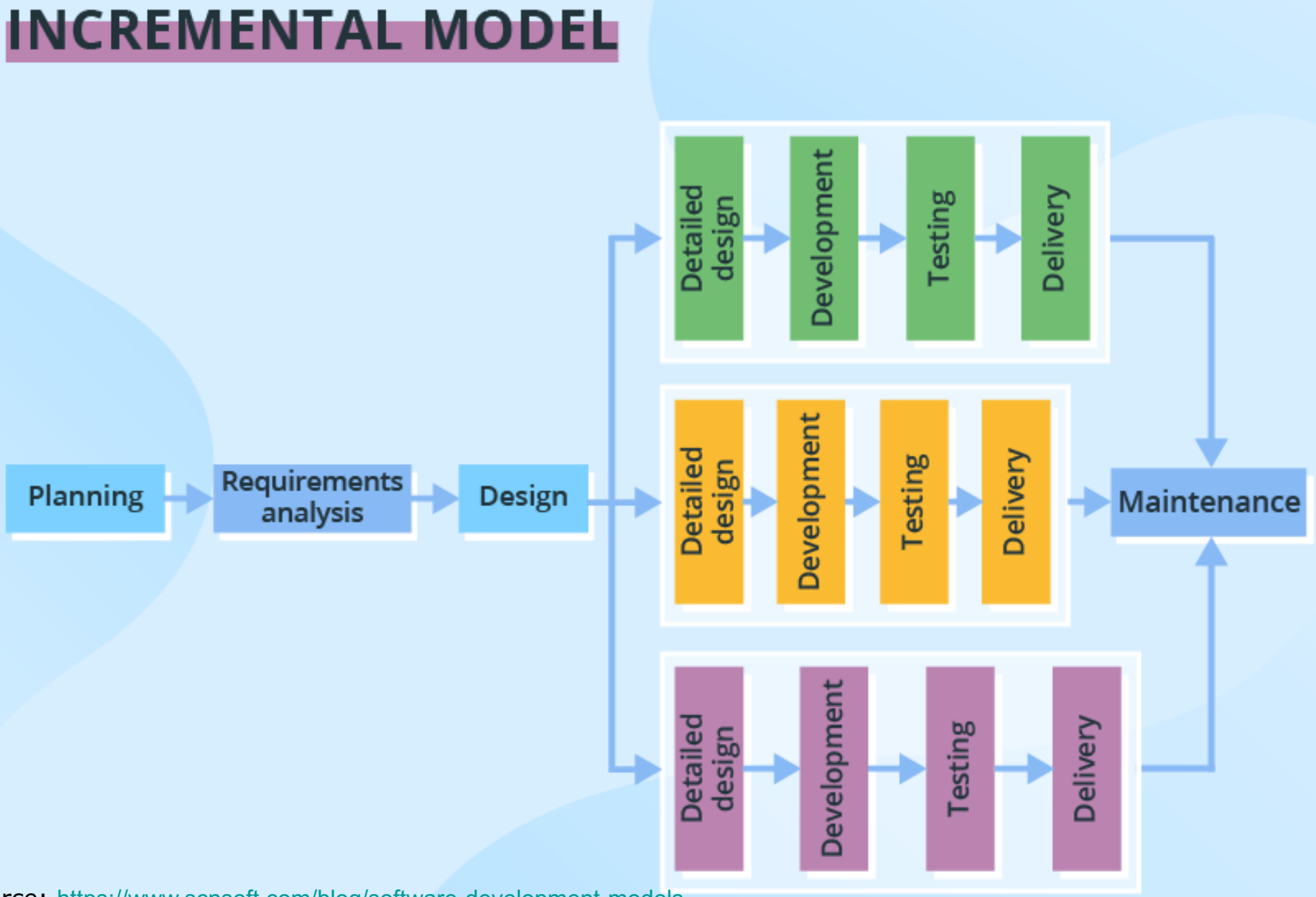
Source: https://www.scnsoft.com/blog/software-development-models

V-Model



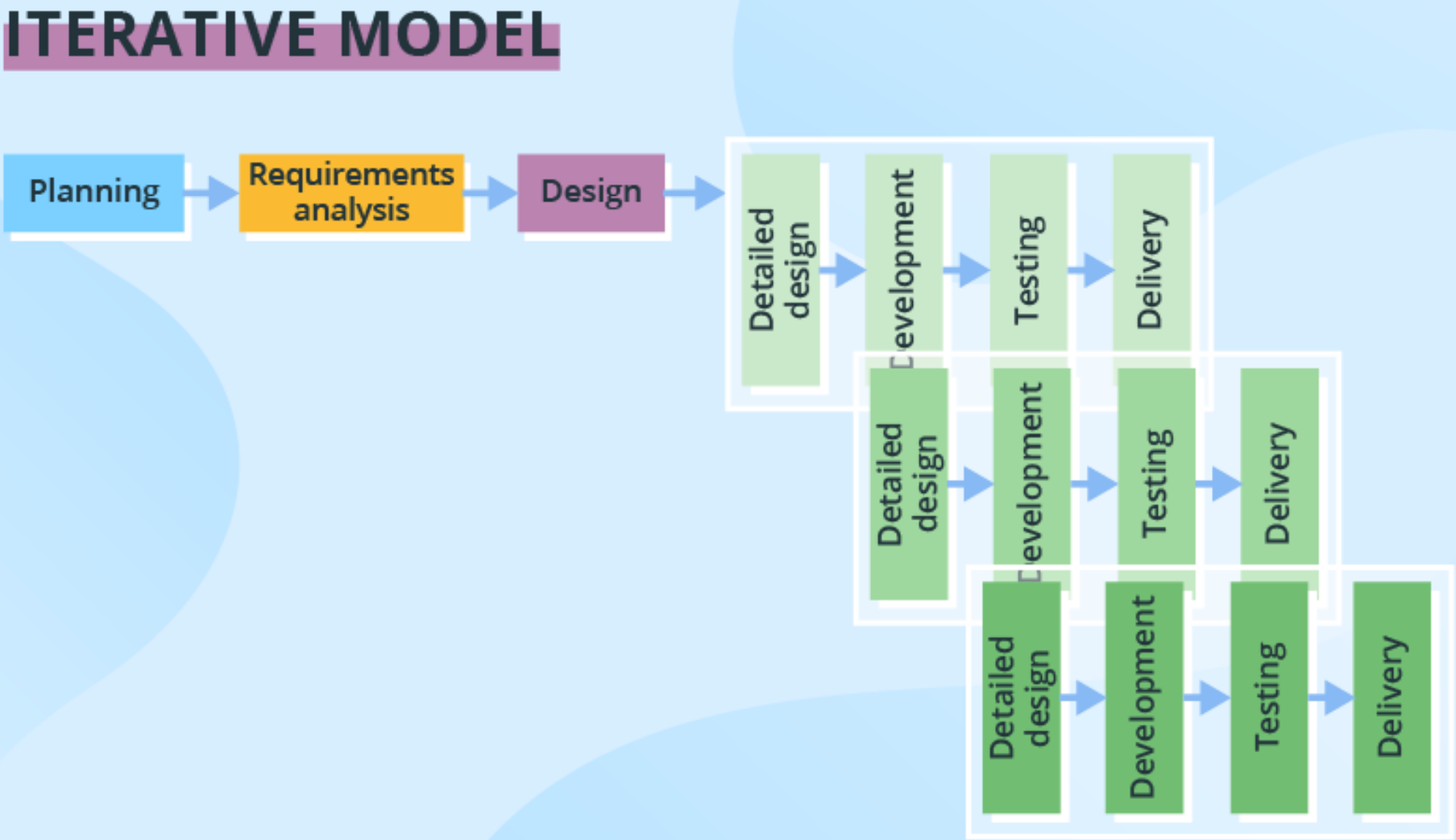
Source: https://www.scnsoft.com/blog/software-development-models

Incremental Model



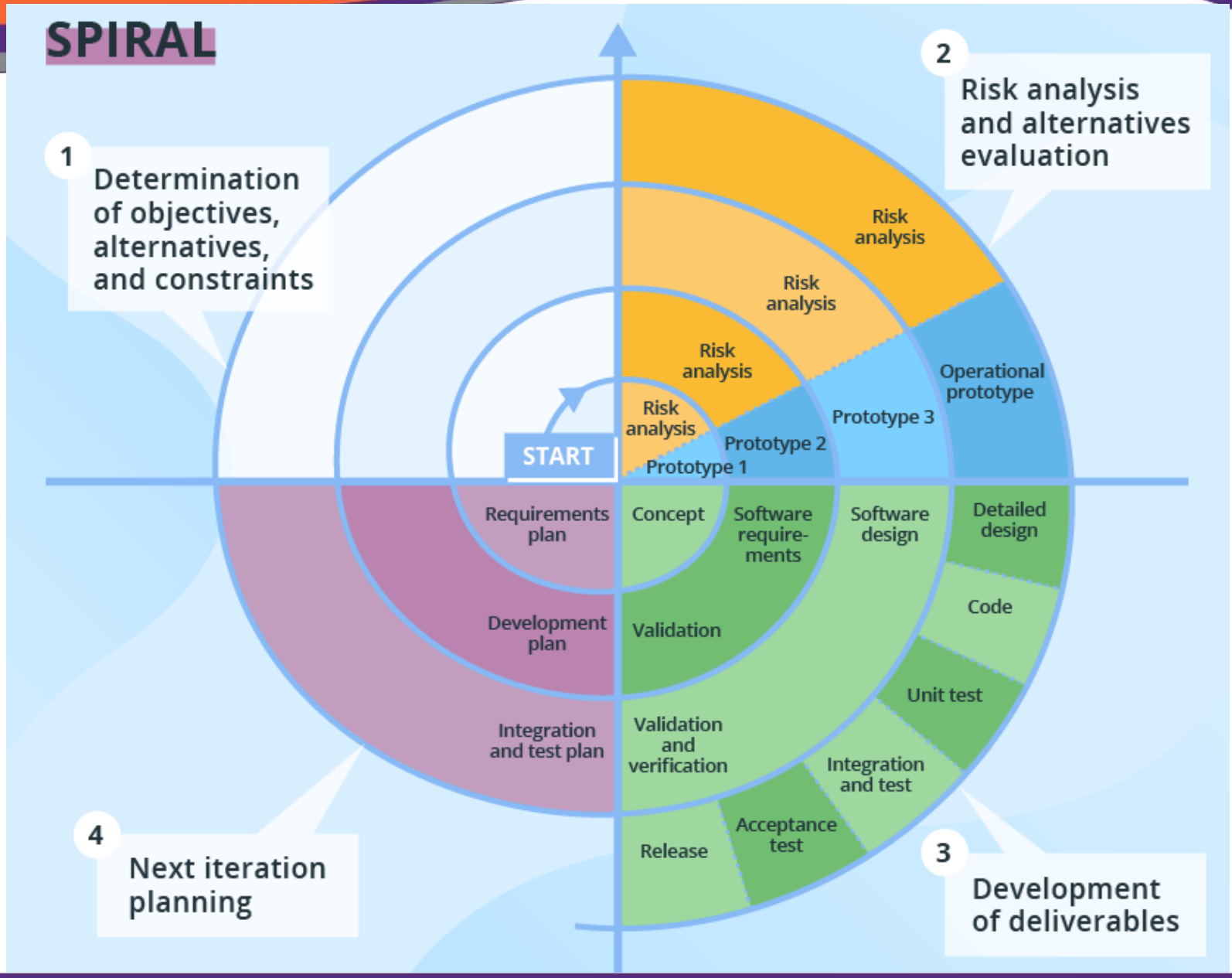
Source: https://www.scnsoft.com/blog/software-development-models

Iterative Model



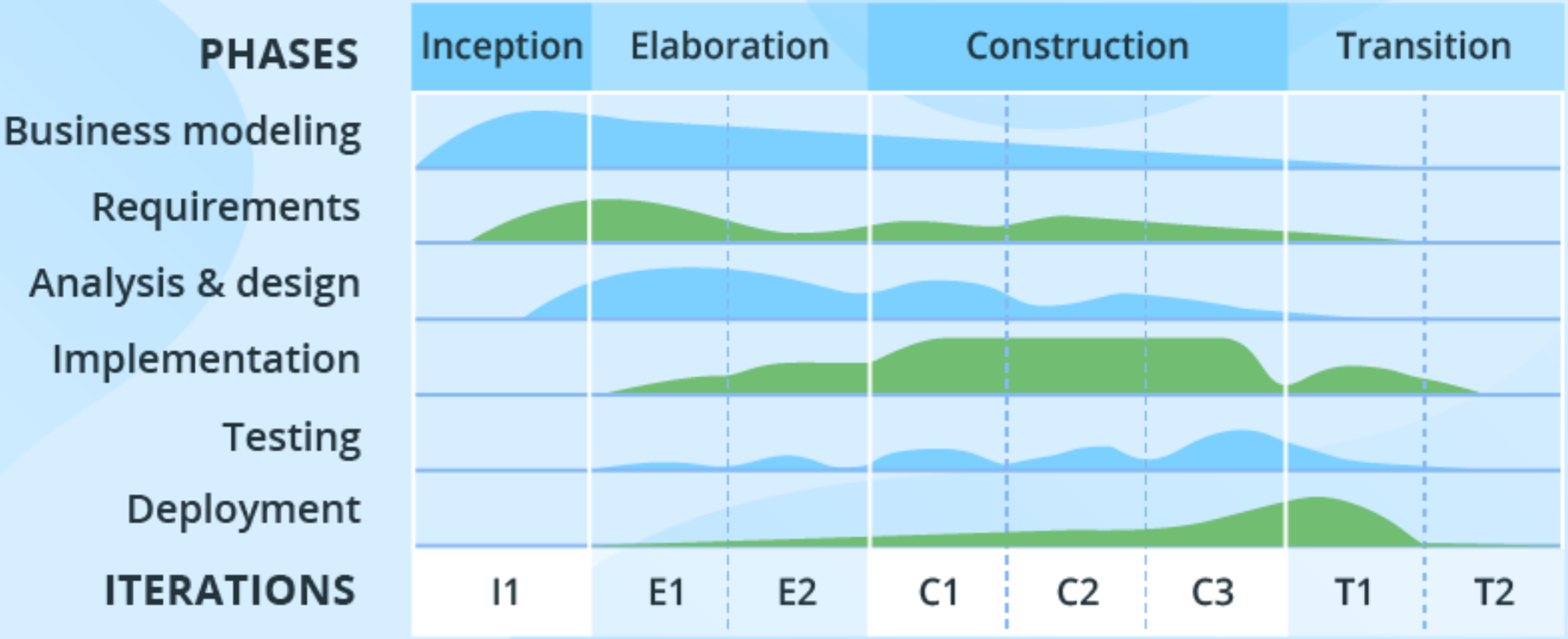
Source: https://www.scnsoft.com/blog/software-development-models

Spiral Model

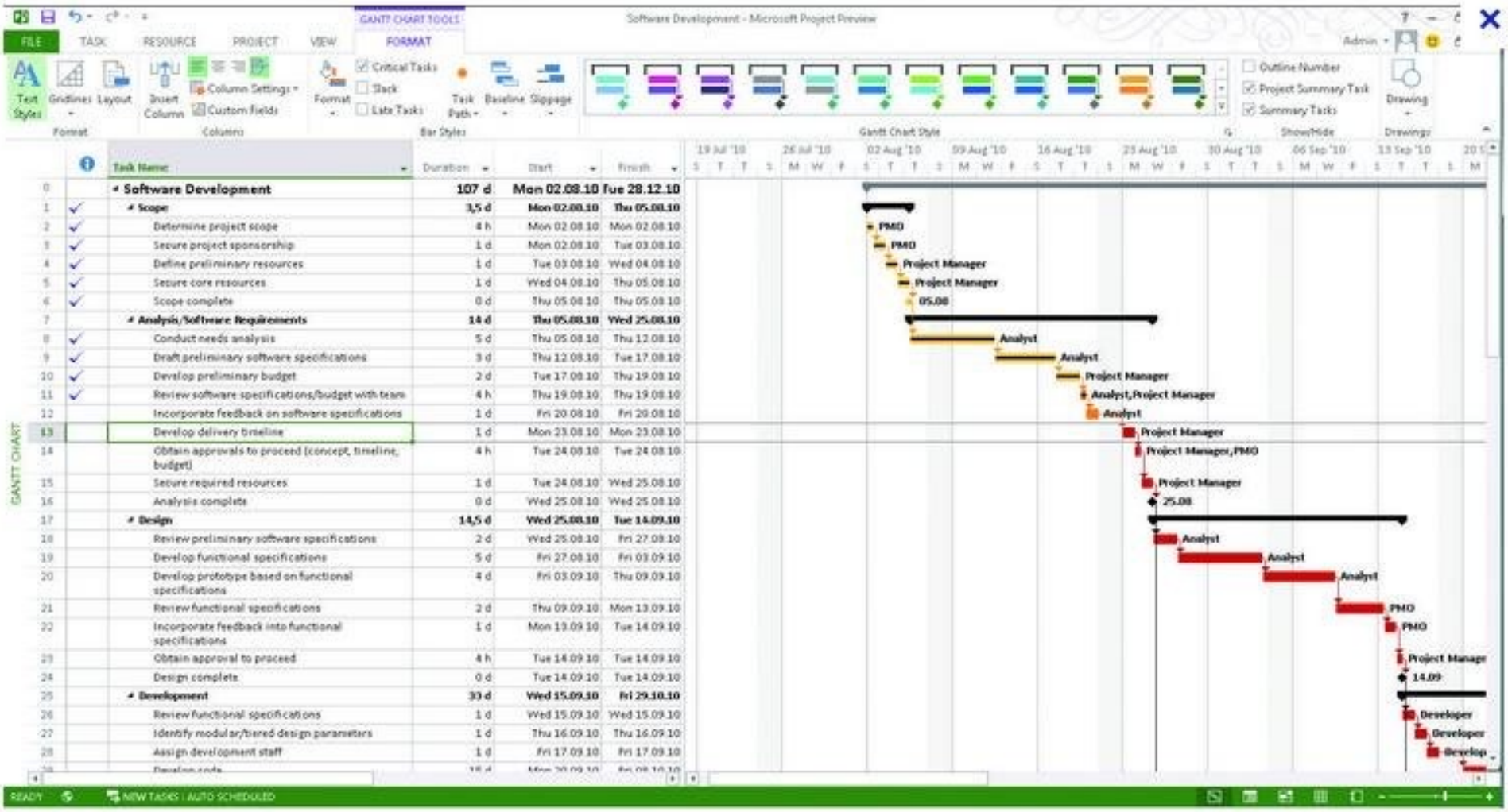


RUP

THE RATIONAL UNIFIED PROCESS (RUP)



Source: https://www.scnsoft.com/blog/software-development-models


Software Development - Microsoft Project Preview
GANTT CHART TOOLS
FILE
TASK
RESOURCE
PROJECT
VIEW
FORMAT
Text Styles
Gridlines
Layout
Insert Column
Column Settings
Custom Fields
Format
Critical Tasks
Slack
Late Tasks
Task Path
Baseline
Slippage
Outline Number
Project Summary Task
Summary Tasks
Drawing
Format
Columns
Bar Styles
Gantt Chart Style
Show/Hide
Drawings
Task Name
Duration
Start
Finish
0 Software Development 107 d Mon 02.08.10 Tue 28.12.10
1 Scope 3,5 d Mon 02.08.10 Thu 05.08.10
2 Determine project scope 4 h Mon 02.08.10 Mon 02.08.10
3 Secure project sponsorship 1 d Mon 02.08.10 Tue 03.08.10
4 Define preliminary resources 1 d Tue 03.08.10 Wed 04.08.10
5 Secure core resources 1 d Wed 04.08.10 Thu 05.08.10
6 Scope complete 0 d Thu 05.08.10 Thu 05.08.10
7 Analysis/Software Requirements 14 d Thu 05.08.10 Wed 25.08.10
8 Conduct needs analysis 5 d Thu 05.08.10 Thu 12.08.10
9 Draft preliminary software specifications 3 d Thu 12.08.10 Tue 17.08.10
10 Develop preliminary budget 2 d Tue 17.08.10 Thu 19.08.10
11 Review software specifications/budget with team 4 h Thu 19.08.10 Thu 19.08.10
12 Incorporate feedback on software specifications 1 d Fri 20.08.10 Fri 20.08.10
13 Develop delivery timeline 1 d Mon 23.08.10 Mon 23.08.10
14 Obtain approvals to proceed (concept, timeline, budget) 4 h Tue 24.08.10 Tue 24.08.10
15 Secure required resources 1 d Tue 24.08.10 Wed 25.08.10
16 Analysis complete 0 d Wed 25.08.10 Wed 25.08.10
17 Design 14,5 d Wed 25.08.10 Tue 14.09.10
18 Review preliminary software specifications 2 d Wed 25.08.10 Fri 27.08.10
19 Develop functional specifications 5 d Fri 27.08.10 Fri 03.09.10
20 Develop prototype based on functional specifications 4 d Fri 03.09.10 Thu 09.09.10
21 Review functional specifications 2 d Thu 09.09.10 Mon 13.09.10
22 Incorporate feedback into functional specifications 1 d Mon 13.09.10 Tue 14.09.10
23 Obtain approval to proceed 4 h Tue 14.09.10 Tue 14.09.10
24 Design complete 0 d Tue 14.09.10 Tue 14.09.10
25 Development 33 d Wed 15.09.10 Fri 29.10.10
26 Review functional specifications 1 d Wed 15.09.10 Wed 15.09.10
27 Identify modular/tiered design parameters 1 d Thu 16.09.10 Thu 16.09.10
28 Assign development staff 1 d Fri 17.09.10 Fri 17.09.10
19 Jul '10
26 Jul '10
02 Aug '10
09 Aug '10
16 Aug '10
23 Aug '10
30 Aug '10
06 Sep '10
13 Sep '10
PMO
PMO
Project Manager
Project Manager
05.08
Analyst
Analyst
Project Manager
Analyst,Project Manager
Analyst
Project Manager
Project Manager,PMO
Project Manager
25.08
Analyst
Analyst
Analyst
PMO
PMO
Project Manage
14.09
Developer
Developer
Develop
GANTT CHART
READY
NEW TASKS : AUTO SCHEDULED

Problems

What are some of the issues with the waterfall or other very formal software development processes?

Prof Taylor's Rule of Software Engineering

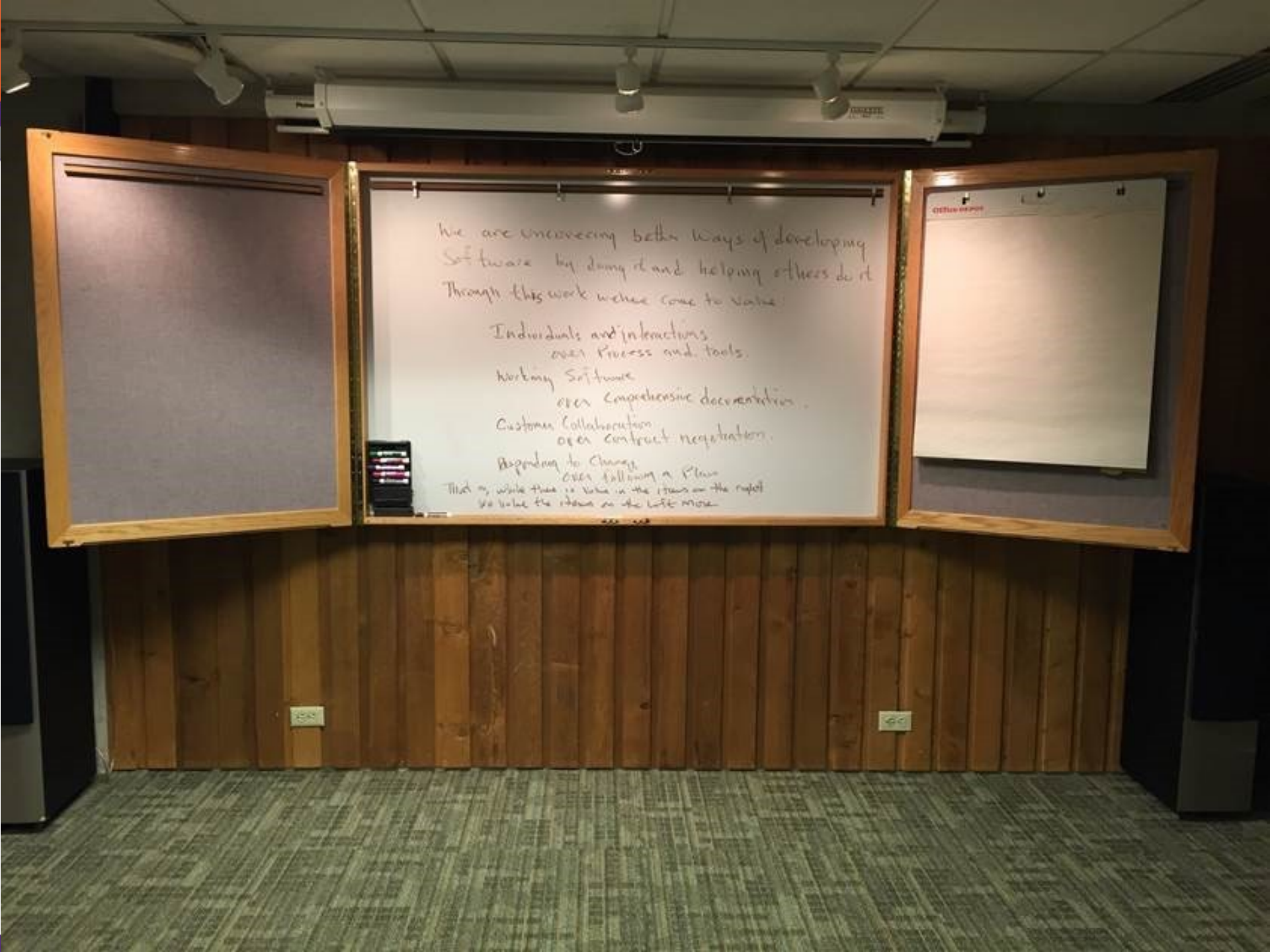
"Software engineering IS requirements discovery"

Process concerns

Classical methods of software development have many disadvantages:

- huge effort during the planning phase
- poor requirements conversion in a rapidly changing environment
- treatment of staff as a factor of production

- Kent Beck, who co-created eXtreme Programming (XP)
- Mike Beedle, co-author of Agile Software Development with Scrum
- Arie van Bennekum, owner of Integrated Agile
- Alistair Cockburn, IT strategist and creator of the Crystal Agile Methodology
- Ward Cunningham, inventor of wiki and first to coin term technical debt
- Martin Fowler, software practitioner, and partner at Thoughtworks
- James Grenning, author of Test-Driven Development
- Jim Highsmith, creator of Adaptive Software Development (ASD)
- Andrew Hunt, co-author of *The Pragmatic Programmer*
- Ron Jeffries, co-creator of eXtreme Programming (XP)
- Jon Kern, who still helps organizations with agile today
- Brian Marick, a computer scientist and author of several books on programming
- Robert C. Martin, also known as "Uncle Bob," who consults via Clean Coding
- Steve Mellor, a computer scientist also credited with inventing Object-Oriented System Analysis (OOSA)
- Ken Schwaber, who co-created Scrum with Jeff Sutherland
- Jeff Sutherland, the inventor, and co-creator of Scrum
- Dave Thomas, programmer, and co-author of *The Pragmatic Programmer*


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 Process 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.

The Agile Manifesto

Individuals and interactions	over	Process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

Source: www.agilemanifesto.org

Agile

- The 2001 Agile Manifesto is closest to a definition
  - Includes a set of 12 principles
- Agile methods are considered
  - Lightweight
  - People-based rather than Plan-based
- No single Agile method
  - Scrum
  - XP
  - Kanban
  - Lean

12 Agile 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.
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 behavior accordingly.

https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

CLEMSON UNIVERSITY

Next Up

- More Agile
- Agile Reading Assignment Homework on Canvas (Due tomorrow!)
- **Quiz 1 (Lessons 1-4) Thursday Jan 25– ~15 minutes, closed-note, 26 points - you will need your computer (try the tech test quiz)**