University of Windsor

LECTURE 2 – SOFTWARE DEVELOPMENT LIFECYCLE

Master of Applied Computing

COMP-8117 : Advanced Software Engineering Topics

Dr. Aznam Yacoub – aznam.yacoub@uwindsor.ca
School of Computer Science

SCHEDULE

- Introduction
- From Waterfall to Iterative methodologies
- Agile Methods and Software Craftmanship
- Models
- ISO Standards

REMINDER

- Software Engineering = Set of *scientific* methods, knowledge, tools, processes, procedures to develop entire industrial reliable software from analysis to the its end of life.

- 2 parts : Management of Realization and Maintenance (Evolution)

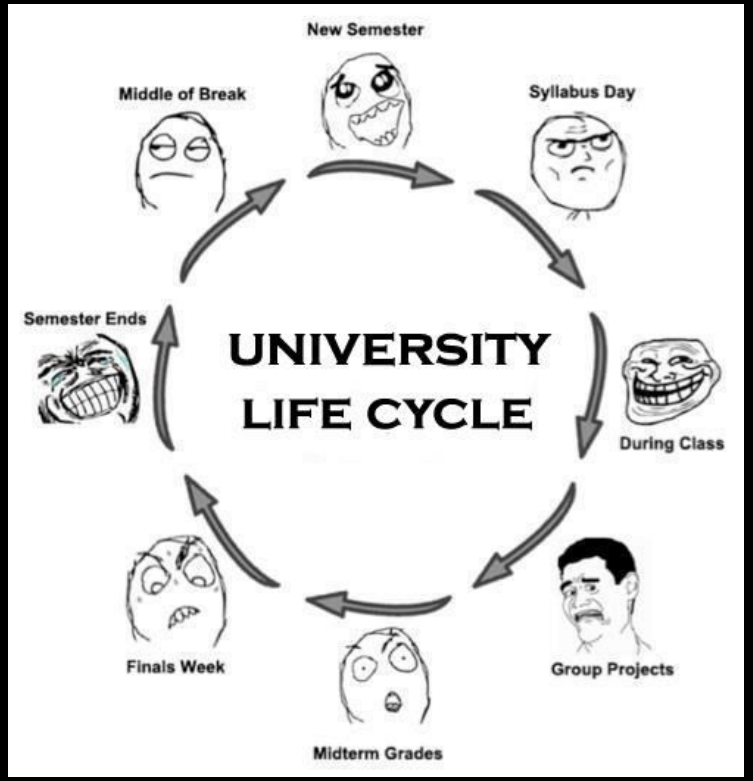
REMINDER

- Software Engineering = Create a set of mathematical computations performed by a computer to solve an informal real life problem.

- Question : How to transform an informal idea into a computational problem and how to transform the computational result into a real-life solution ?

LIFECYCLE CONCEPT

- A software is an object with a finite lifespan.
- Software lifecycle = Natural steps of the life of software from the needs (an idea) to the end of the use of the concrete operationnal product.
- A lifecycle is not a methodology.



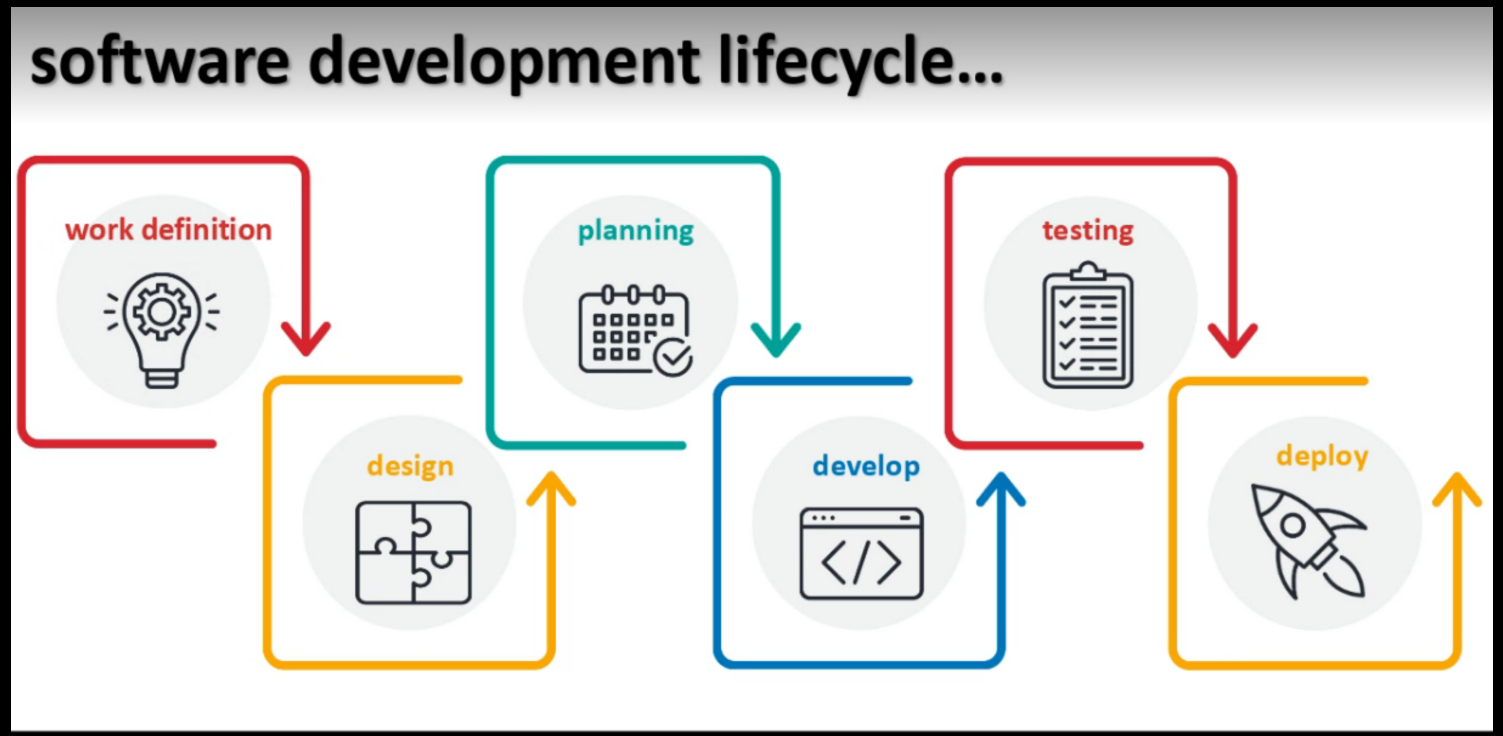
University of Windsor

LIFECYCLE CONCEPT

- Many models of lifecycle.
- 5 minimal essentials steps:
  - Software requirements = Analysis + Specification
  - Software design & *prototyping*
  - Software development / Realization
  - Software testing
  - Software maintenance = production / exploitation / operation

LIFECYCLE CONCEPT

- Other example from Rocket Innovative Studio.



Systems Development Life Cycle (SDLC)
Life-Cycle Phases

Initiation

Begins when a sponsor identifies a need or an opportunity. Concept Proposal is created.

System Concept Development

Defines the scope or boundary of the concept. Includes Systems Boundary Document, Cost Benefit Analysis, Risk Management Plan and Feasibility Study.

Planning

Develops a Project Management Plan and other planning documents. Provides the basis for acquiring the resources needed to achieve a solution.

Requirements Analysis

Analyzes user needs and develops user requirements. Creates a detailed Functional Requirements Document.

Design

Transforms detailed requirements into complete, detailed System Design Document. Focuses on how to deliver the required functionality.

Development

Converts a design into a complete information system. Includes acquiring and installing systems environment; creating and testing databases/ preparing test case procedures; preparing test files; coding, compiling, refining programs; performing test readiness review and procurement activities.

Integration and Test

Demonstrates that the developed system conforms to requirements as specified in the Functional Requirements Document. Conducted by Quality Assurance staff and users. Produces Test Analysis Reports.

Implementation

Includes implementation preparation, implementation of the system into a production environment, and resolution of problems identified in the Integration and Test Phase.

Operations and Maintenance

Describes tasks to operate and maintain information systems in a production environment. includes Post-Implementation and In-Process Reviews.

Disposition

Describes end-of-system activities. emphasis is given to proper preservation of data.

SOFTWARE REQUIREMENT

- Goal : Investigate the problem and requirements, rather than find a solution.
- Find the objects and concepts in the problem domain.
- Software requirements are manageable.

SOFTWARE REQUIREMENT

- Elicitation / Gathering
- Analysis
- Specification
- Validation

ELICITATION

- Example : « I’m responsible of a Formula One Racing Team. I would like to develop a software in which my pilots will be able to train themselves on different tracks in real weather conditions. »

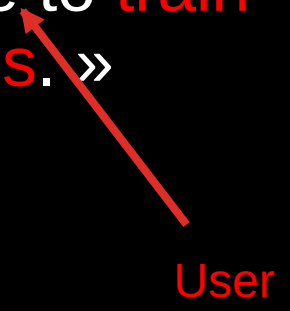
ELICITATION

Domain

- Example : « I'm responsible of a Formula One Racing Team. I would like to develop a software in which my pilots will be able to train themselves on different tracks in real weather conditions. »

ELICITATION

- Example : « I'm responsible of a Formula One Racing Team. I would like to develop a software in which my pilots will be able to train themselves on different tracks in real weather conditions. »



ELICITATION

- Example : « I'm responsible of a Formula One Racing Team. I would like to develop a software in which my pilots will be able to train themselves on different tracks in real weather conditions. »

Main need

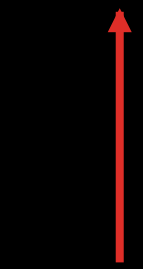
ELICITATION

- Example : « I'm responsible of a Formula One Racing Team. I would like to develop a software in which my pilots will be able to train themselves on different tracks in real weather conditions. »

Application Domain

ELICITATION

- Example : « I'm responsible of a Formula One Racing Team. I would like to develop a software in which my pilots will be able to train themselves on different tracks in real weather conditions. »



Constraint

ELICITATION

- The analysis helps you to understand:
  - What is the customer
  - What is its need
  - What is the primary goal of the product
  - What are the users
  - What are the constraints and main functionnalities

ELICITATION

- The analysis comes before the specification phase and guide it.
- It’s the result of preliminar discussions with the customer.
- Tools : Interviews, questionnaires, user observation, brainstorming, roleplaying and prototyping.

ELICITATION

- Different approaches (Sommerville 1997, Goldsmith 2004, Alexander 2009):
  - Identify the real problem and the technical feasability
  - Identify people who will help to specify requirements
  - Define the technical environment
  - Identify the domain constraints
  - Define an elicitation method
  - Create usage scenarios

ELICITATION

- Different approaches (Sommerville 1997, Goldsmith 2004, Alexander 2009):
  - Identify the REAL problem
  - Identify measures which show the problem is real
  - Identify measures which show the problem has been addressed
  - Identify the causes of the problem, not the problem directly
  - Specify a product design how to satisfy the business requirements

SPECIFICATION

- Describe the software to be developed.
- Defines both functional and non-functional (technical) requirements.
- Provide a basis for estamating costs, risks and schedules.

SPECIFICATION

- Different possible layouts (and some are standardized).
- Example:
  - Purpose > Definitions; Background; Overview; References
  - Overall description > Product perspective (interfaces); Design constraints; Product functions; User characteristics; Technical constraints;
  - Specific requirements > External interface requirements; Performance Evaluation; Functional requirements; Environment Characteristics.

SPECIFICATION

- Software Specification Requirements depend on the domain.
- Goals:
  - Make easier reviews
  - Describe scope of work
  - Provide a reference to software designers
  - Provides use cases for tests
- Software specifications may be formal or non-formal, include mathematical definitions or algorithms.

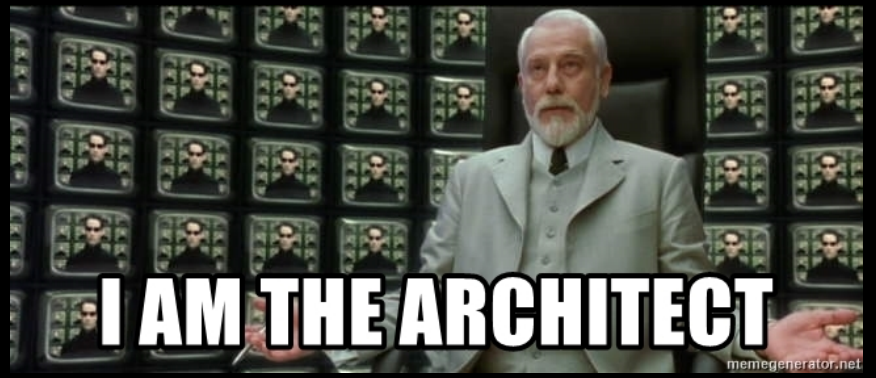
SOFTWARE DESIGN

- Goal : Find a ***conceptual*** solution (software and hardware) that fulfills the requirements, instead of an implementation

- Define software objects and how they collaborate to fulfill the requirements (attributes and methods)

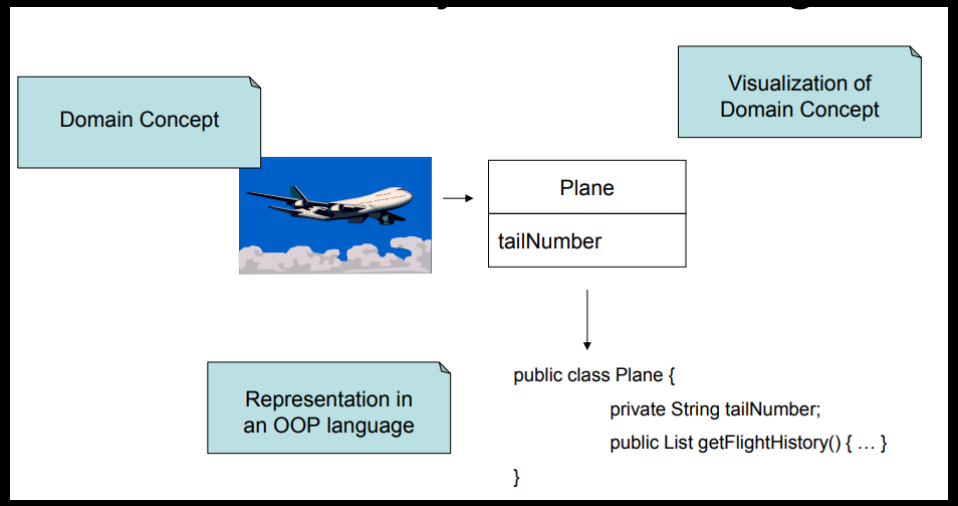
SOFTWARE DESIGN

- Transform a specification to a software artifact that accomplish goals, using a set of constrained components (Ralph and Wand, 2009)
- Define the technical architecture.



SOFTWARE DESIGN : TWO LEVELS

- Simplified design : Translate the requirements into high-level design (represents the user domain) expressed in the language of the technical domain



SOFTWARE DESIGN : TWO LEVELS

- Detailed design : Include the implementation constraints (language, framework, etc.) in low-level model
- Guide the programmer and developer.
- Too detailed design = a lot of constraints
- Not enough detailed design = too much freedom

=> Find the good balance.

SOFTWARE DESIGN

- Fundamental concepts:
  - Abstract / Refinement
  - Modularity
  - Software Architecture
  - Structural Partitionning
  - Data Structure
  - Information hiding

SOFTWARE TESTING

- 2 main activities:
  - Verification : Are we building the software right ?
  - Internal checking against design and requirements (algorithms)
  - Static or dynamic

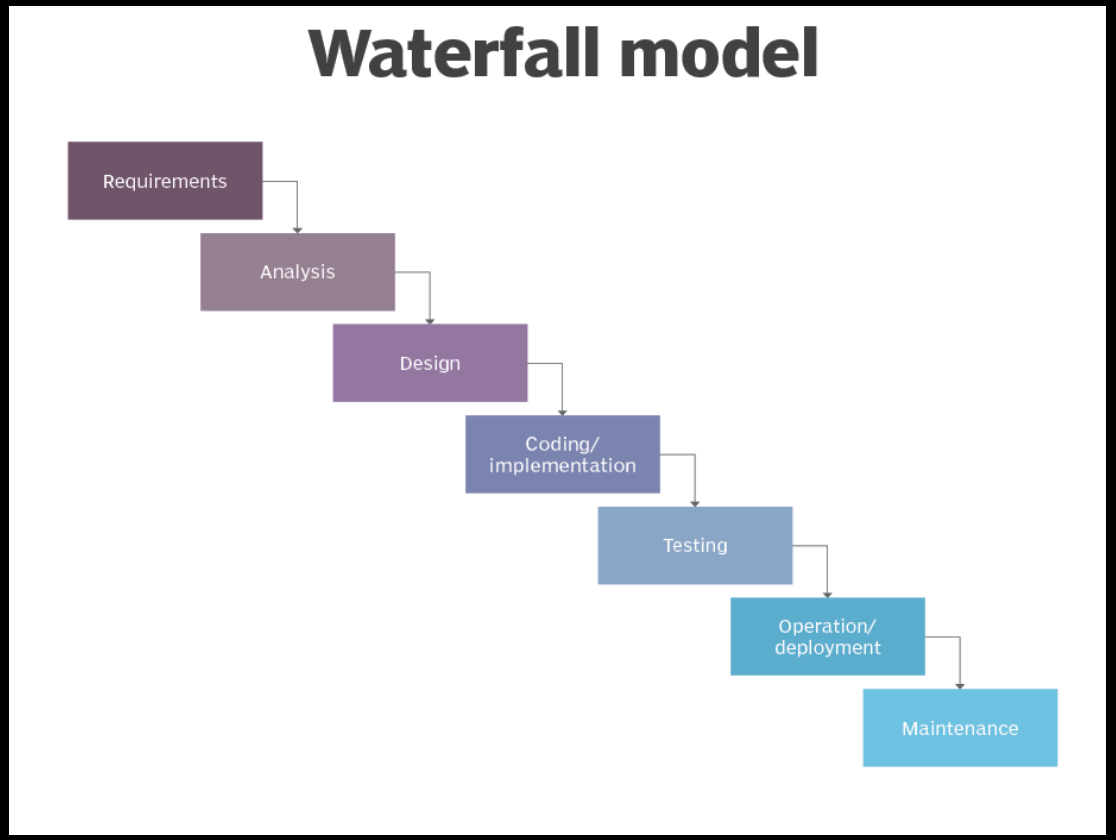
SOFTWARE TESTING

- 2 main activities:
  - Validation : Are we building the right software ?
  - Internal checking against the user needs
  - Dynamic

WATERFALL LIFECYCLE

- Sequential Process
- Philosophy : Define almost everything before programming

WATERFALL LIFECYCLE



WATERFALL LIFECYCLE

- Suitable for « one-shot project » (projects which won’t evolve)
- Suitable for *small* project where requirements and scope are fixed (the customer knows exactly what he wants – *it never happens*)

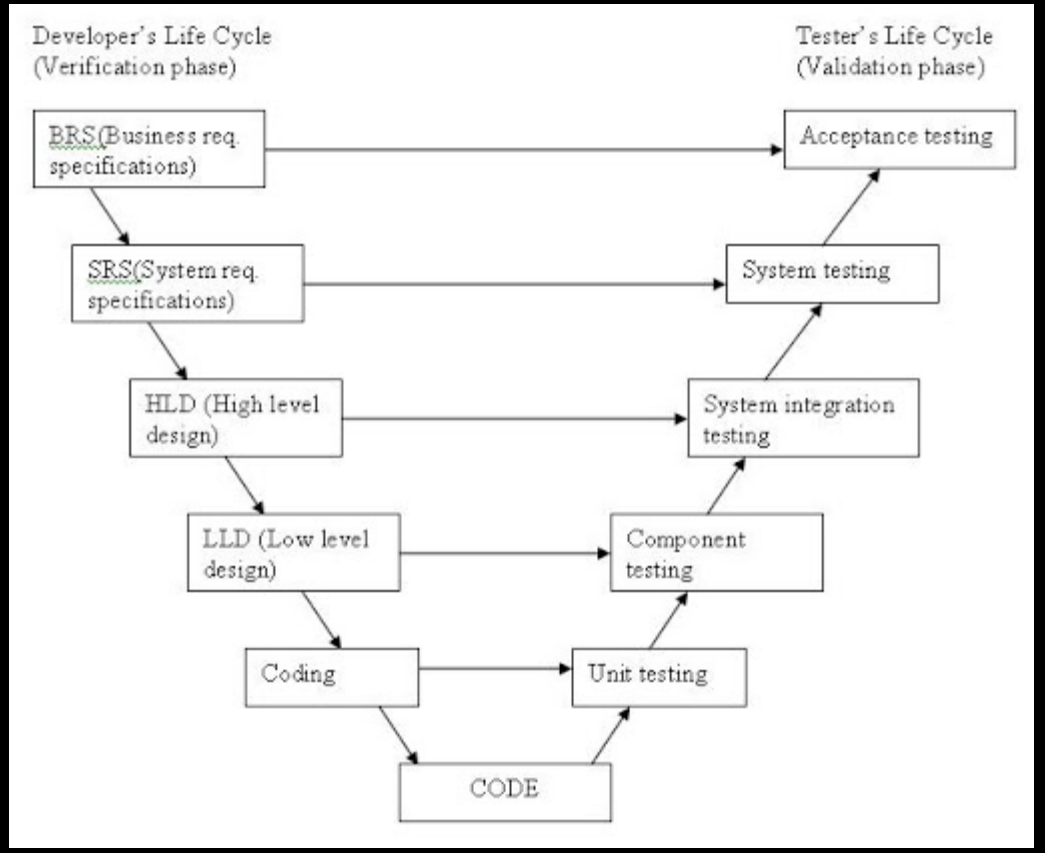
WATERFALL LIFECYCLE

- Problems :
  - High rates of failure
  - A lot of features defined in requirements phase are never used/implemented
  - Waterfall schedules vary up from the final actuals
  - You cannot « return back » to the previous step
  - Tests are defined at the end

V MODEL

- An extension of the waterfall model
- Each phase of the development life cycle is associated to a phase of testing
- Tests scripts are written at the beginning (during the associated phase of the development cycle)
- Adopted in medical device industry

V MODEL



University of Windsor

V MODEL

- Reproduce the same problems that the waterfall model:
  - Linear
  - Testing happen at the end of development
  - Testers are encouraged to look for what they expect to find, rather than exploratory testing.

FEEDBACK AND ADAPTATION

- In complex and changing systems, feedbacks are the key
  - From early development
  - From tests and developers to refine the design models
  - From the progress of the team tackling early features to refine the schedule and estimates
  - From the client and marketplace to re-prioritize the features to develop as soon as possible

FEEDBACK AND ADAPTATION

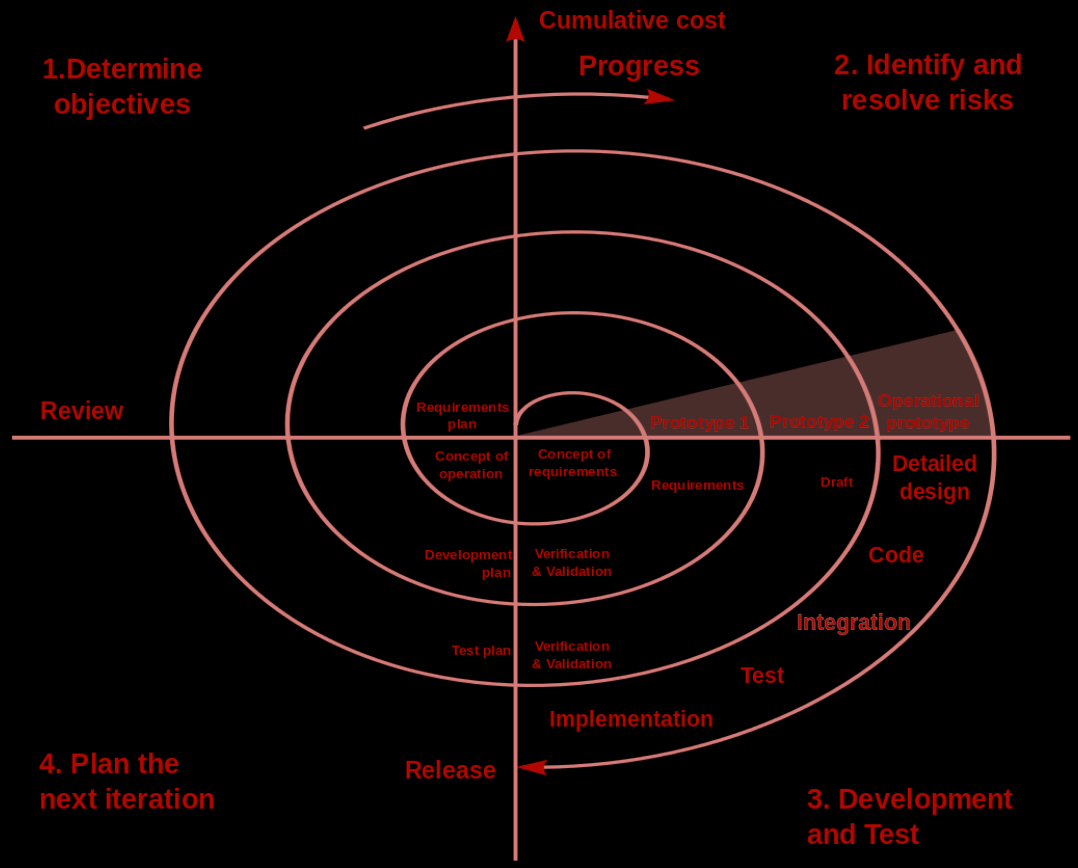
- In complex and changing systems, feedbacks are the key
  - From early development
  - From tests and developers to refine the design models
  - From the progress of the team tackling early features to refine the schedule and estimates
  - From the client and marketplace to re-prioritize the features to develop as soon as possible

=> Risk-driven and Client-driven planning

ITERATIVE AND EVOLUATIONARY LIFECYCLE

- Involves early programming and testing of a partial system, in repeating cycles (iterations) based on increments
- The development starts before all requirements are gathered
- Feedback is used to clarify and improve requirements
- Short development step, feedback, and adaptation to clarify the specifications and design

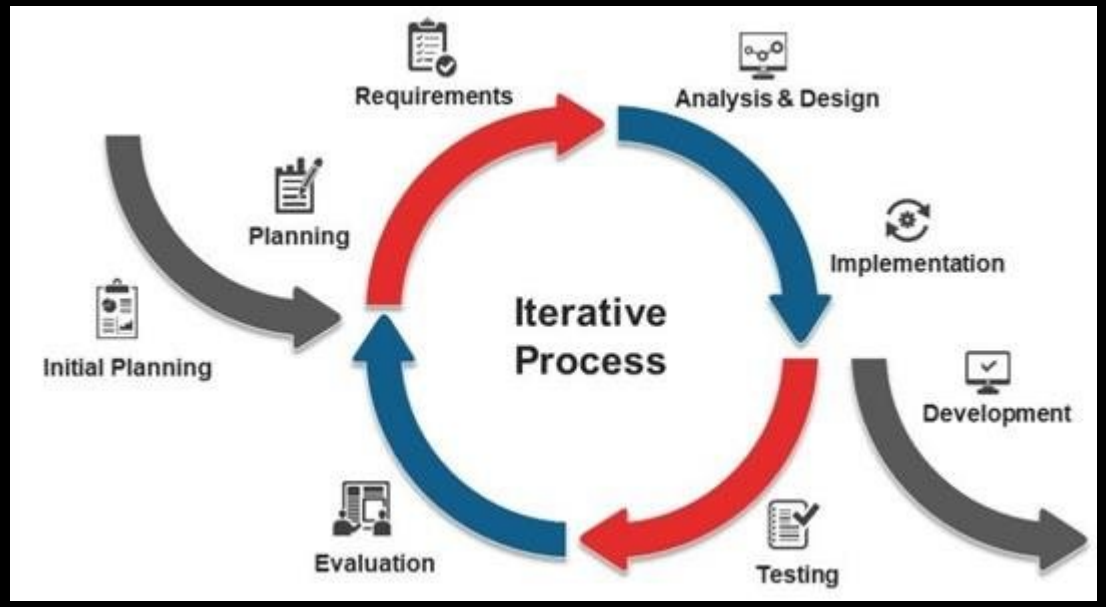
SPIRAL MODEL



ITERATIVE PROCESS

- Development is organized into a series of fixed-length iterations (timeboxed)
- Outcome = tested, integrated and executable incomplete partial system
- Each iteration includes the five activities of software engineering

ITERATIVE PROCESS



ITERATIVE PROCESS

- Successive increment and refinement of a system through multiple iterations
- Result = Production deployment after many iterations
- Output of an iteration is NOT an experimental throw-away prototype. Iterative processes are not prototyping.

ITERATIVE PROCESS

- Philosophy : Change and adaptation are unavoidable and should drive the development (no frozen requirement)
- Less project failure, lower defect rates
- Early mitigation of high risks
- Early visible progress
- Early feedback
- Complexity management
- Methodical learning to improve development one iteration at a time

ITERATIVE PROCESS

- Rules :
  - Iteration length are fixed (set at the beginning of the project)
  - Date slippage is illegal
  - What cannot be completed should be added to future iterations
- Goals :
  - Identify and drive down the highest risks
  - Build visible features that the client cares most about

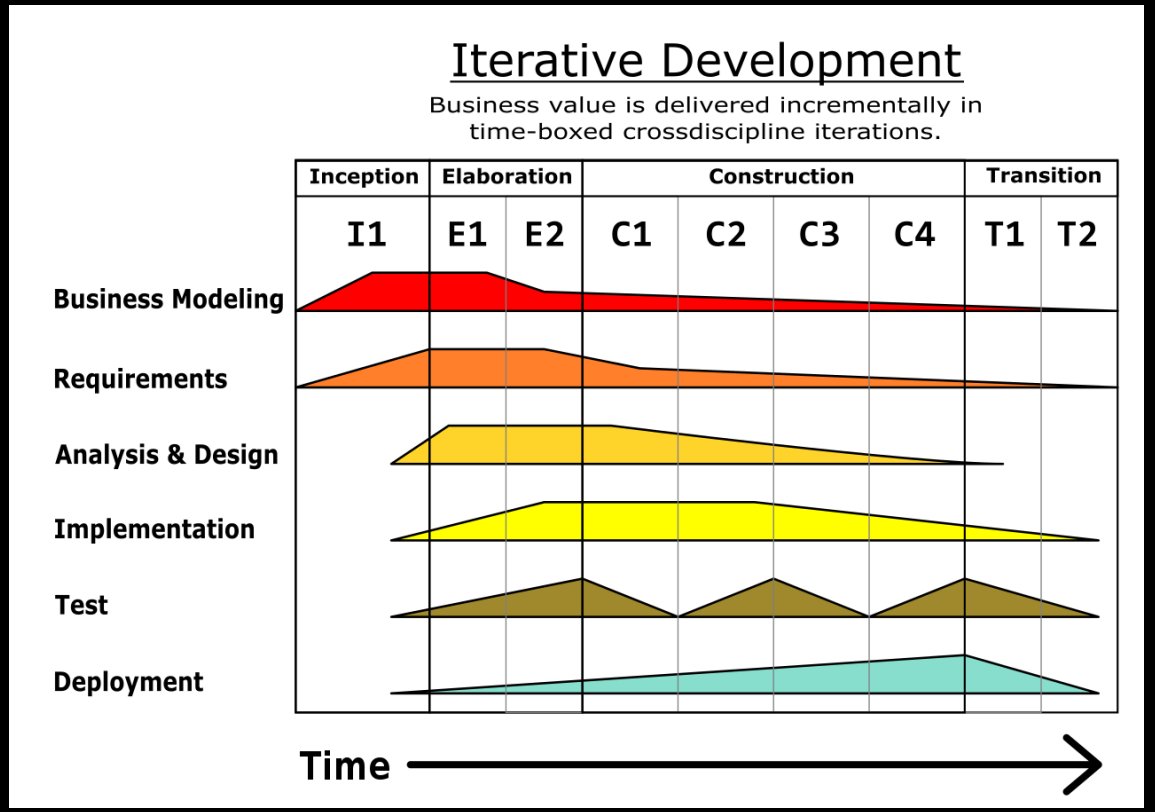
ITERATIVE PROCESS

- Architecture-centric development:
  - First iterations = stabilizing the core architecture
  - Not having a solid architecture is a common high-risk

ITERATIVE PROCESS

- Example of frameworks: (Rational) Unified Process (UP)
- 4 major phases in UP (Jacobson 1999):
  - Inception : Approximate vision, business case, scope, vague estimates
  - Elaboration : refined vision, iterative implementation of the core architecture, resolution of high risks
  - Construction : Iterative implementation of the remaining lower risk, preparation for deployment
  - Transition : beta, deployment

UNIFIED PROCESS



University of Windsor

UNIFIED PROCESS

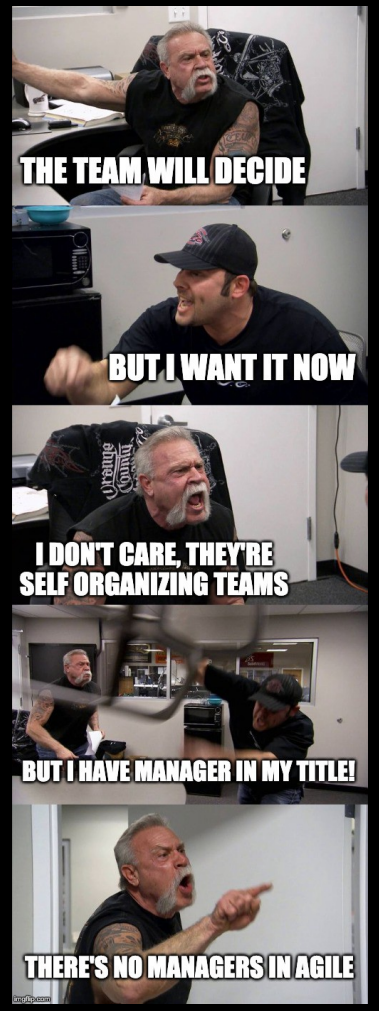
- Disciplines = set of activities in one software engineering area
  - For example, within requirement analysis : business modeling, requirements, design
- UP implementation = building the system, not deploying it
- Development case = document describing the practices and UP artifacts for a project
- UP = Good balance between need and stability (vs reactive to feature creep)

UNIFIED PROCESS

- Disciplines = set of activities in one software engineering area
  - For example, within requirement analysis : business modeling, requirements, design
- UP implementation = building the system, not deploying it
- Development case = document describing the practices and UP artifacts for a project
- UP = Good balance between need and stability (vs reactive to feature creep)
- UP = flexible, can be applied in lightweight and agile approach

AGILE DEVELOPMENT

- Agile is NOT do what you want, when you want.
- Agile methods (SCRUM, XP, etc.) :
  - Apply timeboxed iteration philosophy
  - Use adaptive planning
  - Promote incremental delivery
  - Include other values and practices that encourage agility (rapid and flexible response to change)



University of Windsor

AGILE DEVELOPMENT

- In Agile methods, every step is iterative
  - Incremental refinement of plans
  - Incremental requirements
  - Incremental design
- Each iteration doesn’t have all the activities
- Put the emphasis on:
  - Self-organization (Individuals and interactions) over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan

AGILE DEVELOPMENT

- Put the emphasis on development, rather than documentation
- But **it doesn't mean:**
  - No documentation
  - No process
  - No contract
  - No plan

- Agile methods are a philosophy reflected in:
  - Space organization (open workspace)
  - Team organization (scrum master instead of project leader)
  - Cookies and cakes

AGILE DOCUMENTATION

- Purpose of documentation is to understand, not to explain everything
- Specification and Design are not avoided, but reduced to the strict necessary
- Documentation support communication
- Don’t model everything
- Use simple tools (whiteboard)
- Create models in parallel and know they are inaccurate

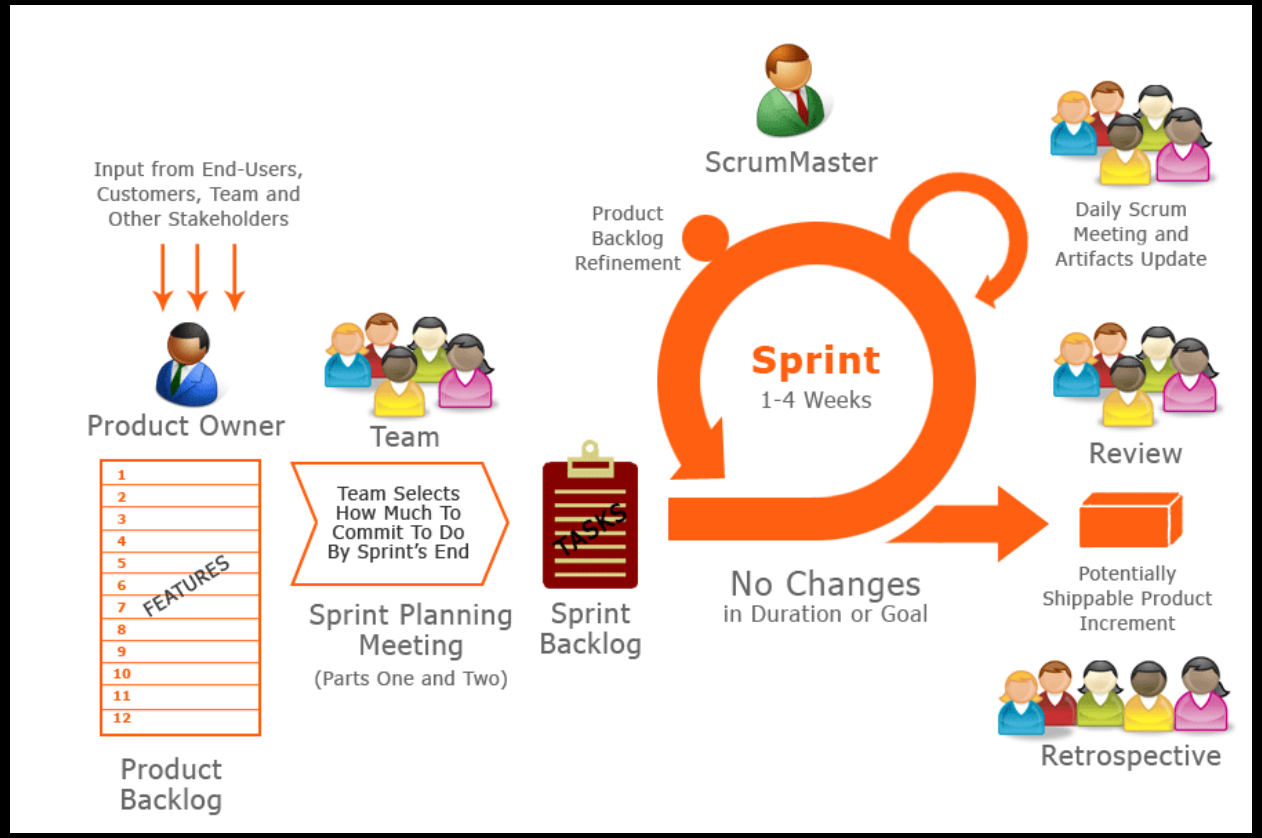
AGILE UNIFIED PROCESS

- Implantation of UP in an Agile Framework
- Prefer small set of UP activities and artifacts
- Requirements and designs are not completed before implementation
- No detailed plan :
  - Phase plan = estimate the project and date and milestones
  - Iteration plan = greater detail of one iteration in advance

OTHER AGILE APPROACHES

- eXtreme Programming (based on test-driven approaches)
- SCRUM
- KANBAN
- Rapid Application development
- Scrumban
- Feature-driven development
- *Water SCRUM Fall*

SCRUM



University of Windsor

SCRUM - DEFINITIONS

- Product owner : represents the customers, responsible of the product backlog; defines the product in customer-centric terms;
- Scrum master : buffer between the team and any distracting influences; the scrum master doest not decide, but it makes the communication easier by verifying that any steps are followed;
- Sprint = Iteration
- Daily scrum : 15 minutes daily meeting in which each developer presents its tasks and results
- Sprint backlog : list of functionnalities to implement during a sprint

SCRUM – COMMON MISTAKES

- A Scrum Master is a team leader => Remember SCRUM promotes self-organization ! NO CHIEF. The Scrum Master is a big brother, who cares about the team, not a dictator.

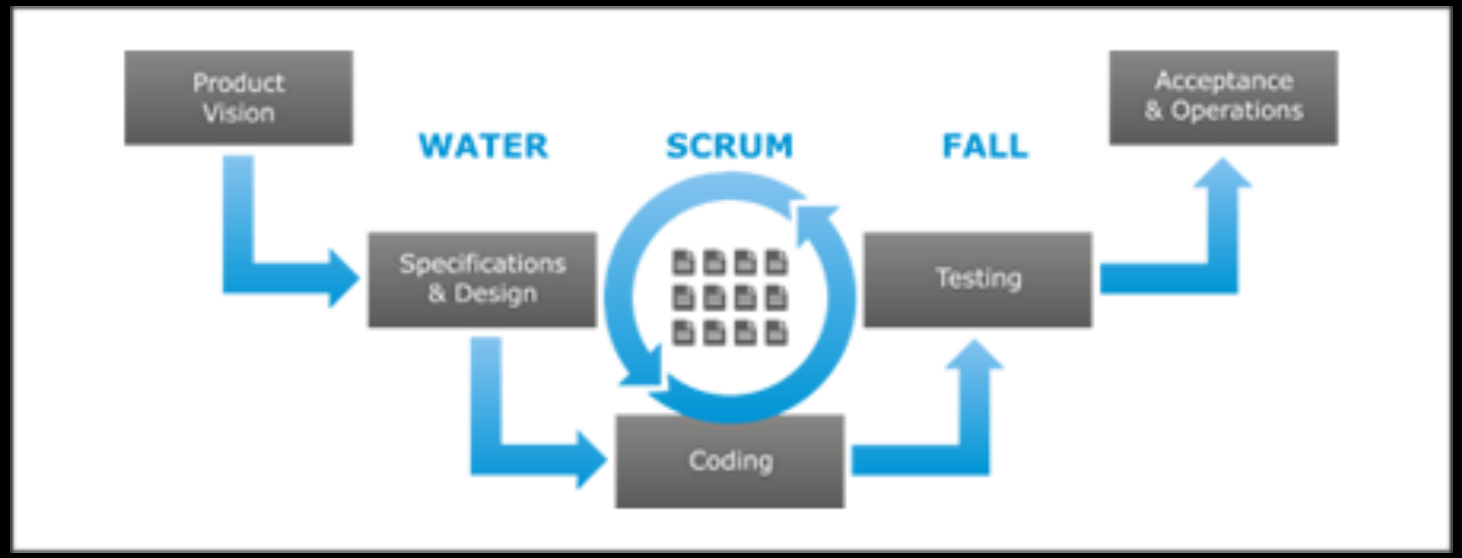
- I can change goals of a sprint => Once a sprint is started, no change in the objectives can occur during this sprint. You can defer a task, but you cannot add or cancel a planned objective.

- A sprint doesn’t necessary product an artifact => Remember that you have to produce something valuable at each sprint (documentation, executable…)

SCRUM – LIMITATIONS

- SCRUM doesn’t work well when people are not in the same room or if a team has full-time and part-time members
- SCRUM doesn’t work well with too specialized members, it was designed for interdisciplinary development and people with T-shaped skills
- SCRUM doesn’t work well in project with a lot of external dependencies
- SCRUM doesn’t work well with project which needs regulated quality control (example: medicale devices, vehicle control) : each increments should be fully developed and tested in a single sprint, which is not the case for products that need a lot of regression testing

WATER SCRUM FALL - EXAMPLE



SOFTWARE CRAFTMANSHIP

- Software development approach which puts the emphasis on the coding skills, rather than financial considerations
- Software developer = rigorous apprentice/practitioner of scientific approach with computatioal theory
- Software engineer = precision, predictability, measurement, risk mitigation, professionalism => codified bodies of knowledge and certification
- Craft (Well-crafted) Software vs Crap (Engineered) Software !

SOFTWARE CRAFTMANSHIP

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

SOFTWARE CRAFTMANSHIP

- Software craftmanship philosophy = « Software development is a *scientific* craft, more than an engineering activity »

- Born from the Agile Manifesto

- Is another form of XP and Scrum :
  - Quality : Simple design, Test-driven development (XP)
  - Humility : I question myself (Scrum)
  - Share : Pair-programming (XP)
  - Pragmatism : Adaptation (Scrum)
  - Professional : My customer is a partner (XP)

SOFTWARE CRAFTMANSHIP

- Agile Methods put the emphasis on doing the right product.
- Software craftmanship puts the emphasis on doing the product right.
- Engineering or scientific approaches only don't ensure you'll do good work. Actually, be a Software Craftengineer.

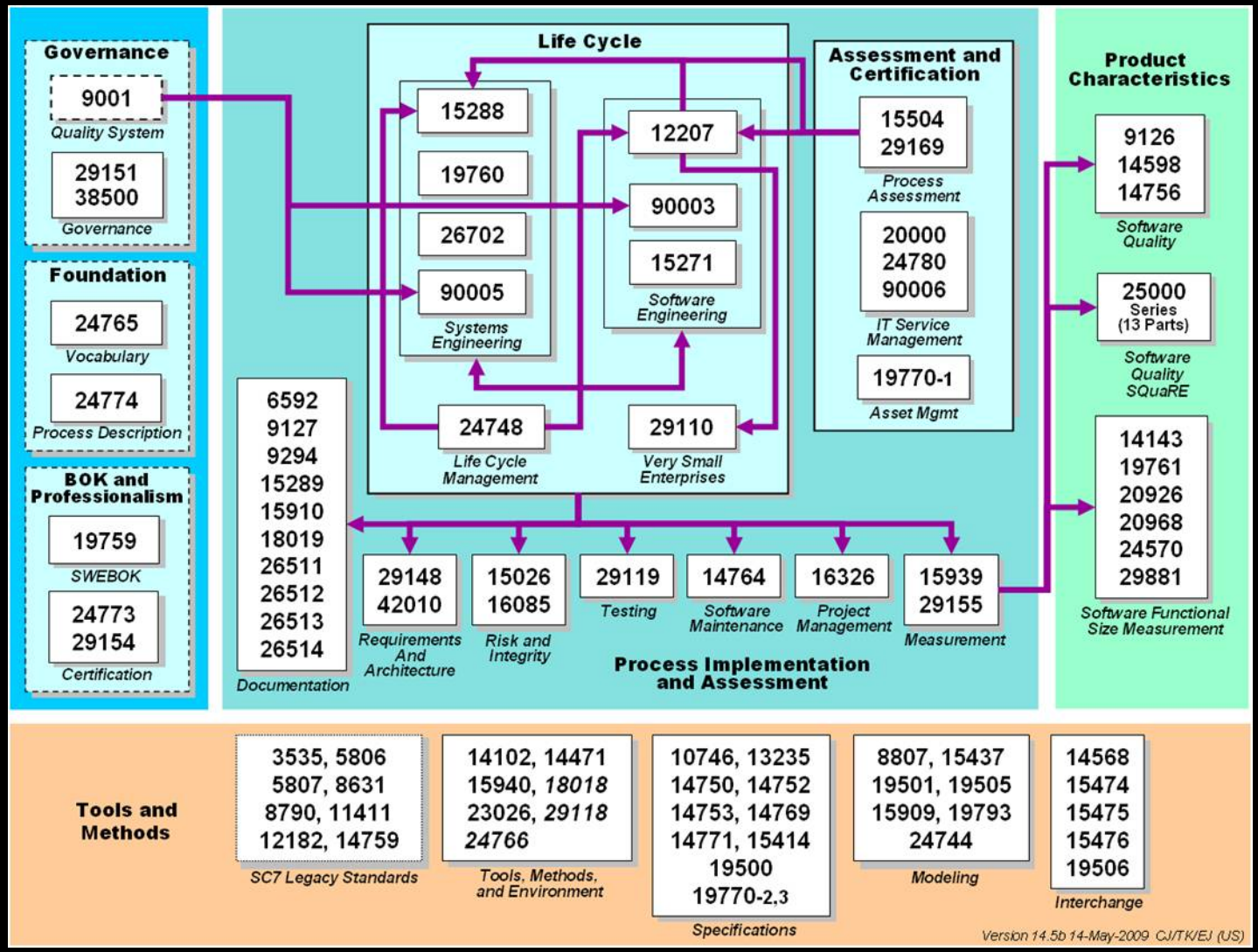
University of Windsor

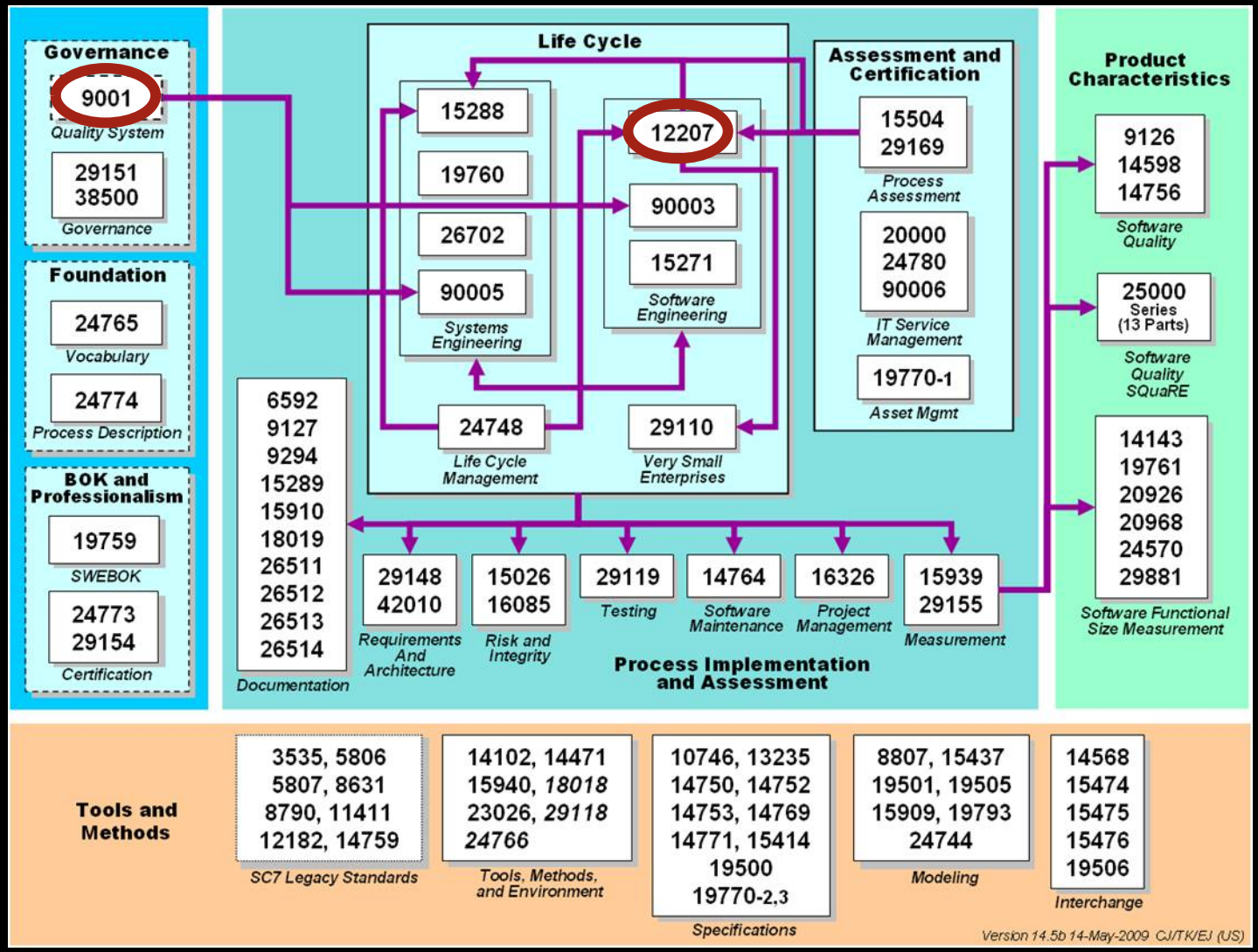
ISO/IEC STANDARDS

- Software Engineering Methods have been standardized by ISO (International Orgnization for Standardization) and EIC (International Electrotechnical Commission)
- Companies following standards are certified => Proof of quality and maturity
- Standards emphasize communication and shared understanding => « Tests are complete. » - What does it mean ?

ISO/IEC STANDARDS

- Benefits:
  - Improved management of software development
  - Visible certification can attract new customers or are sometimes required
  - Enhancement of partnerships and co-development
- Standars:
  - Encapsulation of best practices
  - Framework for quality assurance process
- Problems:
  - Small companies don’t adopt standard – especially « Agile » companies
  - Difficulty to apply standards





ISO-9000

- Family of Standard for quality management
- ISO-9001 : Most important for Software Engineering Companies
- Quality refers to all those features of a product/service required by a customer.
- Quality management = how to ensure that the product satifies customer’s quality requirements + comply with regulations

ISO-9000 - PHILOSOPHY

- Document what you do
- Do what you document
- Record what you did
- Prove it

- Standard on the process, not the products !

ISO-12207

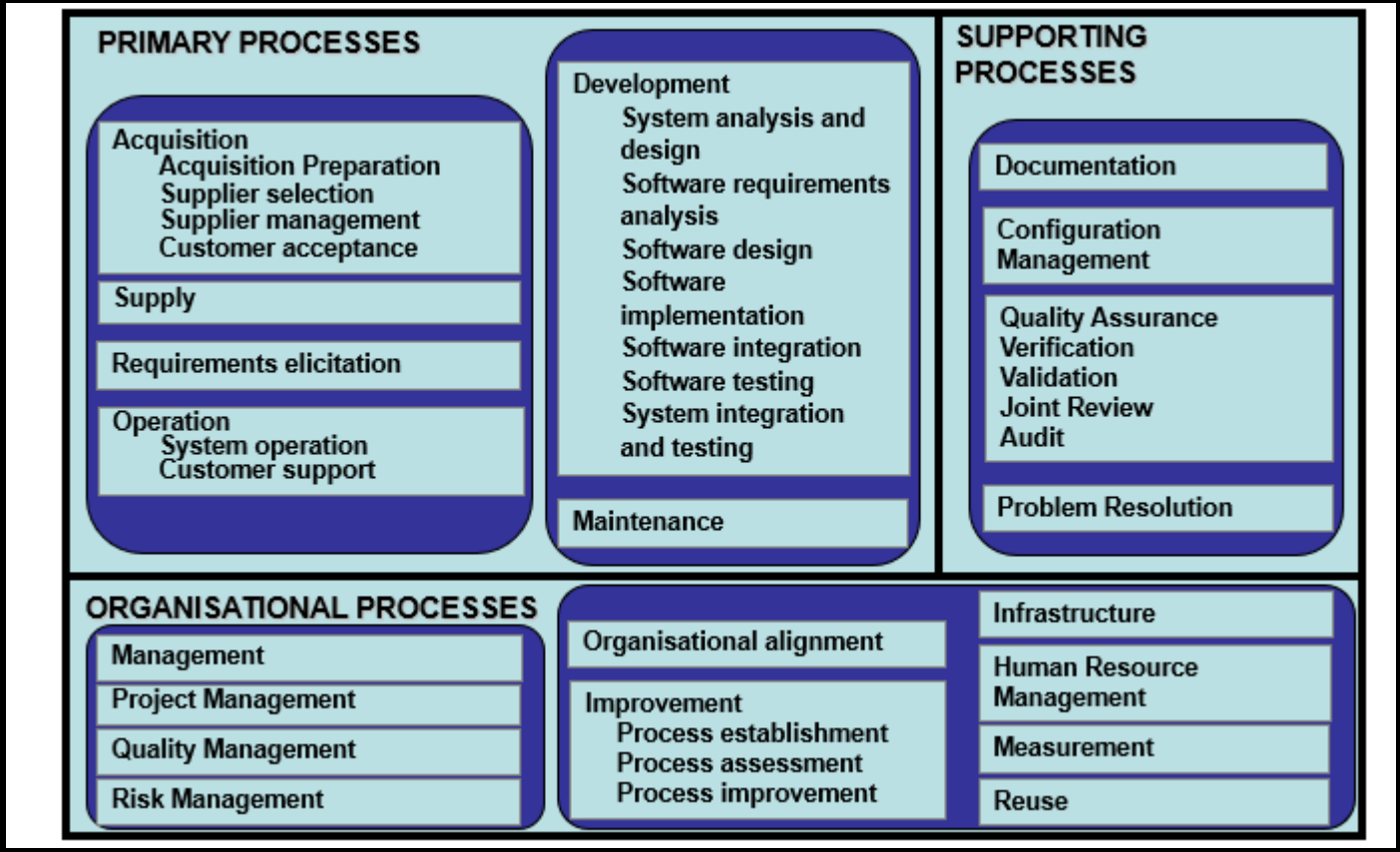
- Defines the software engineering standard processes, tasks and lifecycles
- It's THE standard that defines all the tasks required for developing and maintaining software
- Provides a common framework to speak the same language in software discipline
- What it defines ?
  - High level process architecture
  - Activities and Tasks
  - Inventory of processes from which to choose

ISO-12207

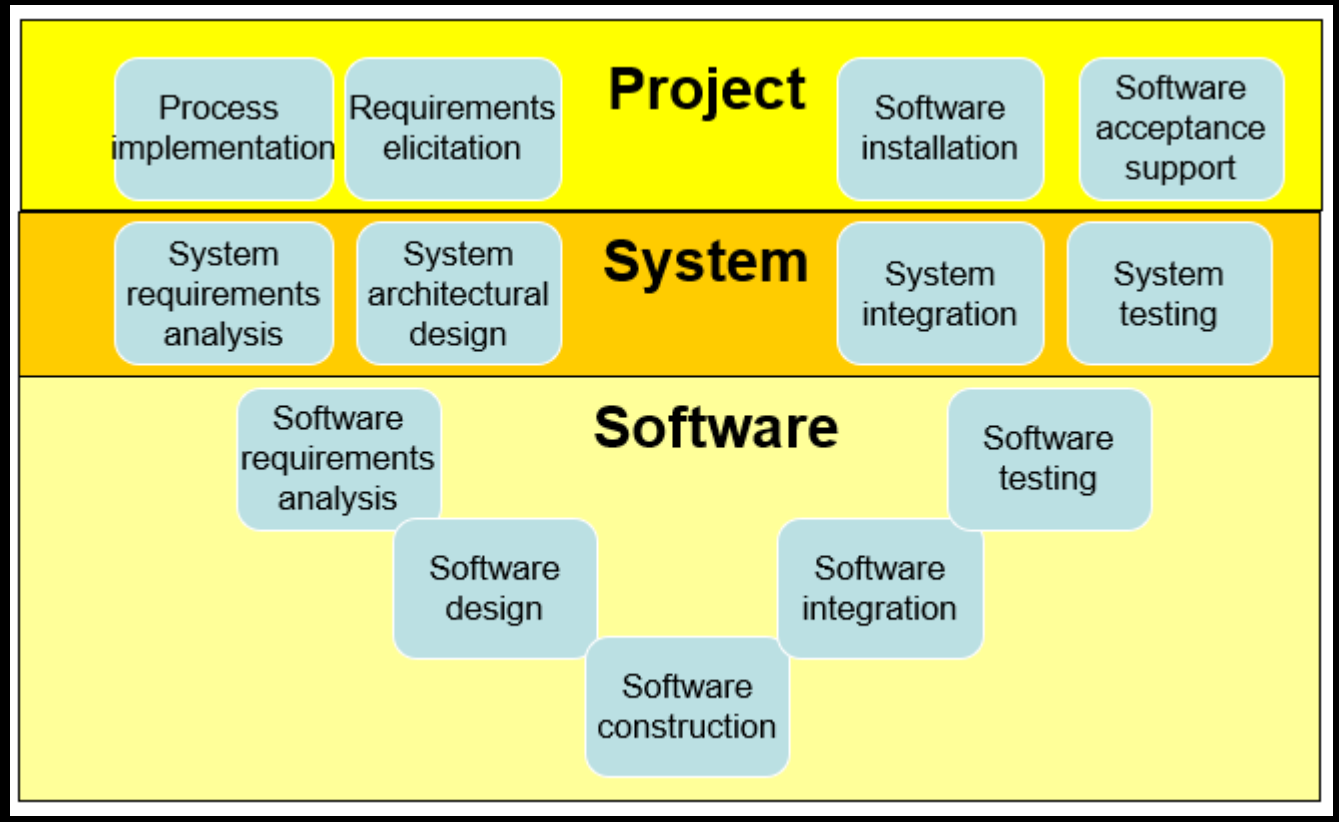
- But it’s a standard for lifecycles, not a standard for methods !
  - It does not prescribe specific lifecycle or specific tools

- According to ISO-12207, there are:
  - 23 processes
  - 95 activities
  - 325 tasks
  - 224 outcomes associated to each process

ISO-12207



ISO-12207



CONCLUSION

- Software Engineering = 5 key activities
- Linear development approaches = good for small projects and domains which require a lot of safety check
- Iterative development approaches = good for big/complex projects and domains with a lot of instability (change of requirements, etc.)
- Standard lifecycles provide framework which have to be setup according to each project

CONCLUSION

- Agile Methods = Well-used modern approaches with emphasis on communication, adaptation and self-organization
- Codified methodologies => You don’t do what you want when you want
- It’s not magic !

REFERENCES

This lecture is based on:

- COMP-8117 (Winter 2020) – Dr. Ziad Kobti
- Software Engineering (Fall 2020) – Dr. Amine Hamri, Dr. Aznam Yacoub
- Software Engineering – Ian Sommerville
- Intro do ISO/IEC SE standards (O'Connor, Dublin City University, Ireland)

