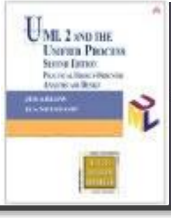


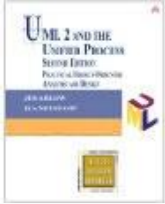
Lecture 4

AGILE AND OTHER METHODS

PB007 Software Engineering I
Faculty of Informatics, Masaryk University

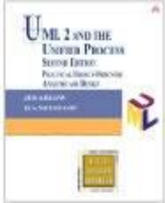
Fall 2020

Outline



- ✧ Software Process Models
- ✧ Agile Development
- ✧ Agile Practices
- ✧ Agile Methods

- ✧ UML State diagram



SOMMERVILLE

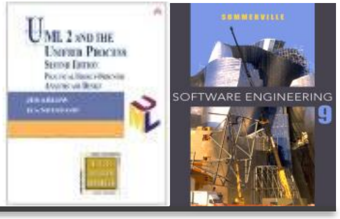
SOFTWARE ENGINEERING

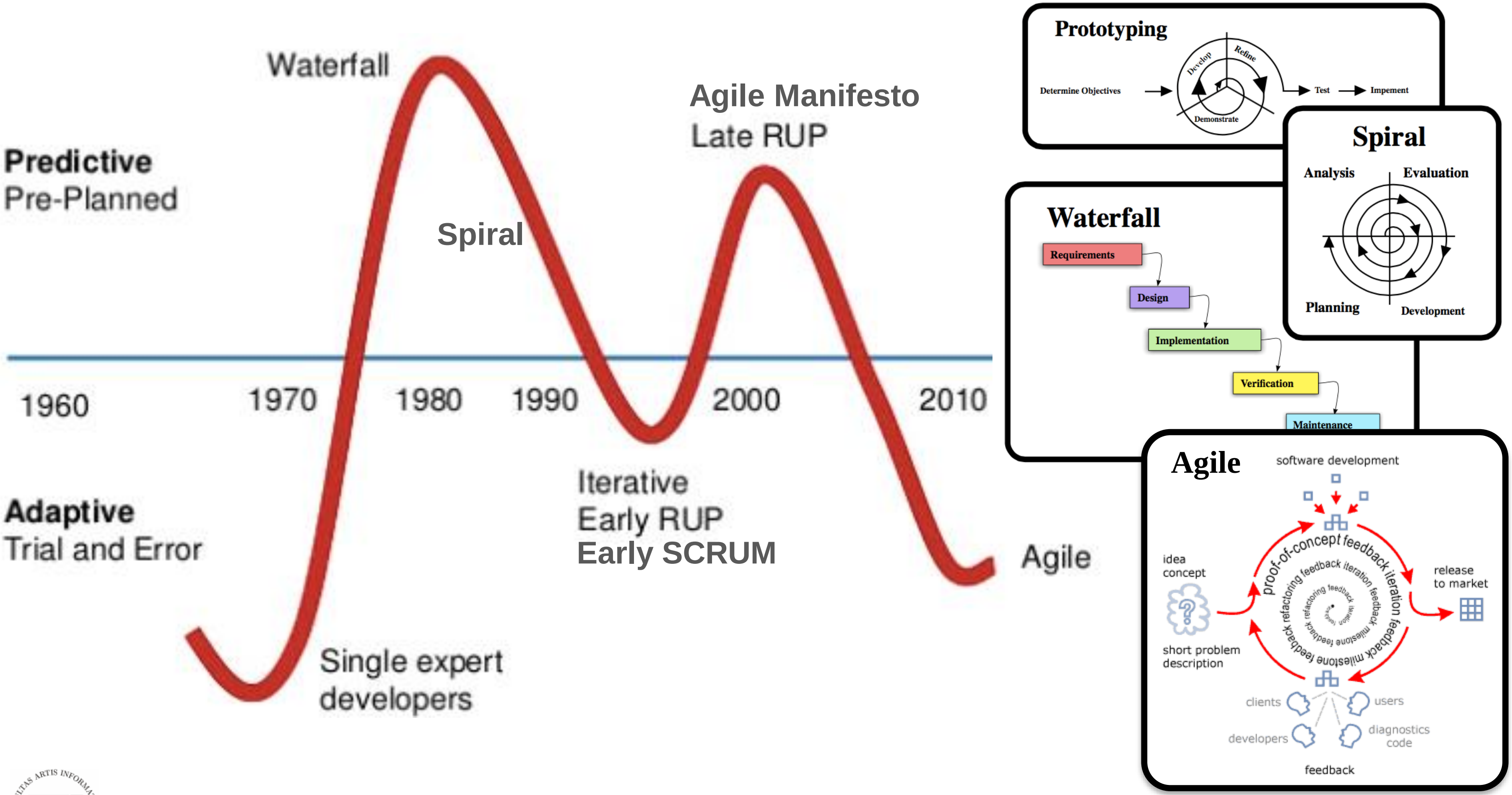
9

Software Process Models

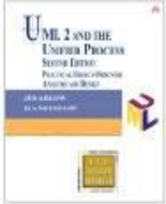
Lecture 4/Part 1

Software process models



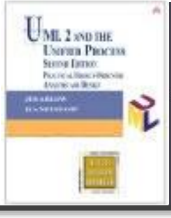


Software process models



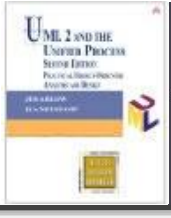
- **The waterfall model**
  - Plan-driven model. Separate and distinct phases of specification and development.
- **Incremental development**
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- **Reuse-oriented software engineering**
  - The system is assembled from existing components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements **from many different models**.

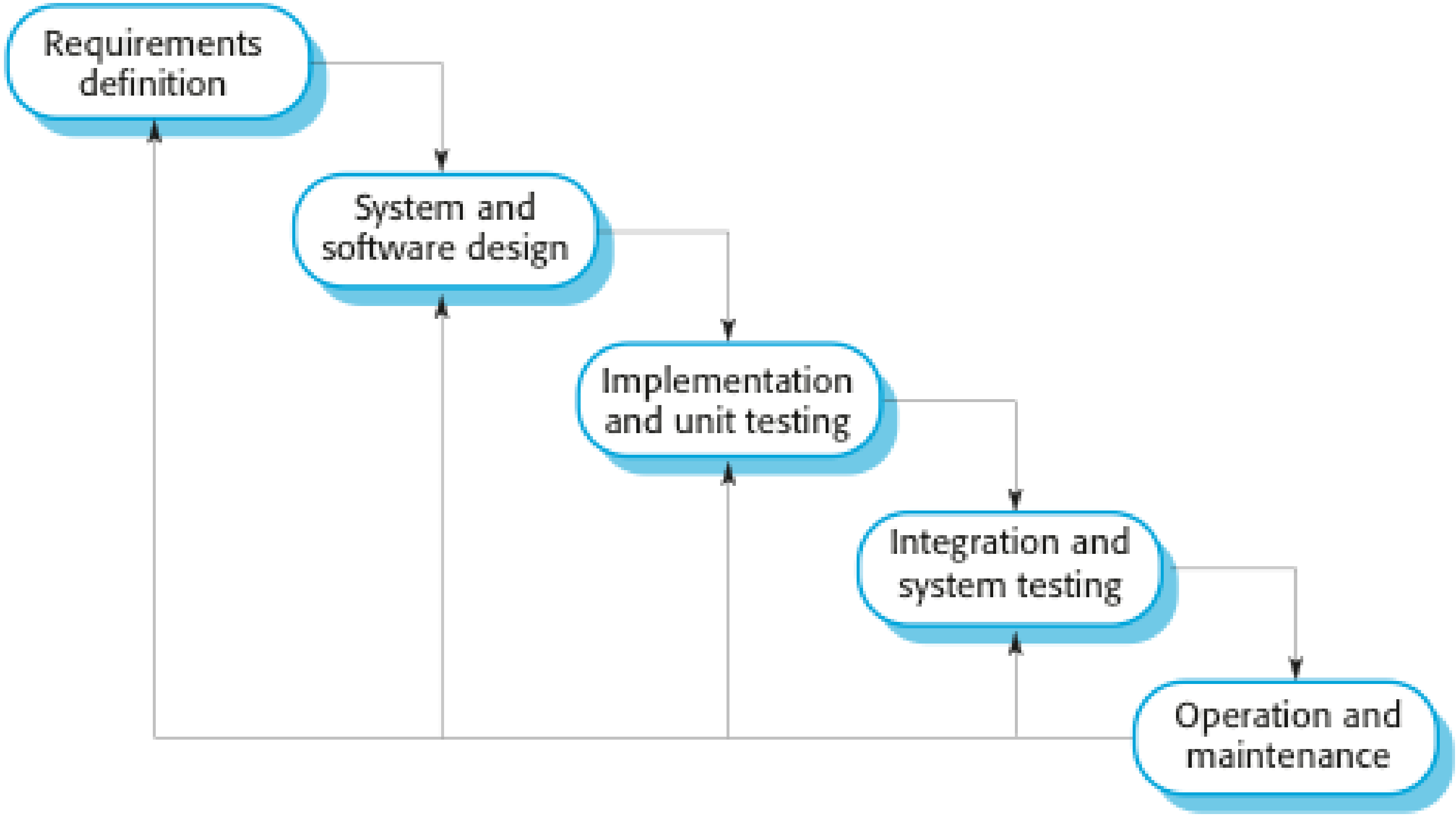
Plan-driven and agile development



- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the **outputs** to be produced at each of these stages **planned in advance**.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the **outputs** from the development process are **decided through a process of negotiation** during the software development process.

The waterfall model




Requirements definition

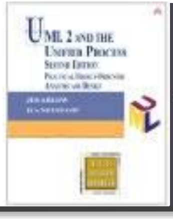
System and software design

Implementation and unit testing

Integration and system testing

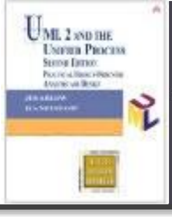
Operation and maintenance

Waterfall model benefits and problems



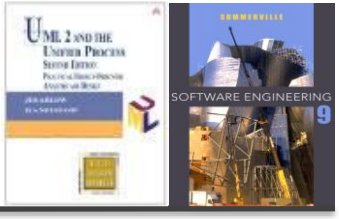
- The waterfall model is mostly used for **large system engineering projects** where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.
- Suitable for new versions of **generic products.**
  - Well understood context, stable requirements.
- The process makes it difficult to respond to **changing customer requirements**.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes can be limited.

Software prototyping



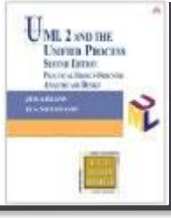
- A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- A prototype can be used in:
  - The requirements engineering process to help with **requirements elicitation**, consistency checking and validation;
  - In design processes to **explore design options** and develop a **UI design**;
- Prototypes often have **poor internal structure** and thus should not become the foundation of the final system.

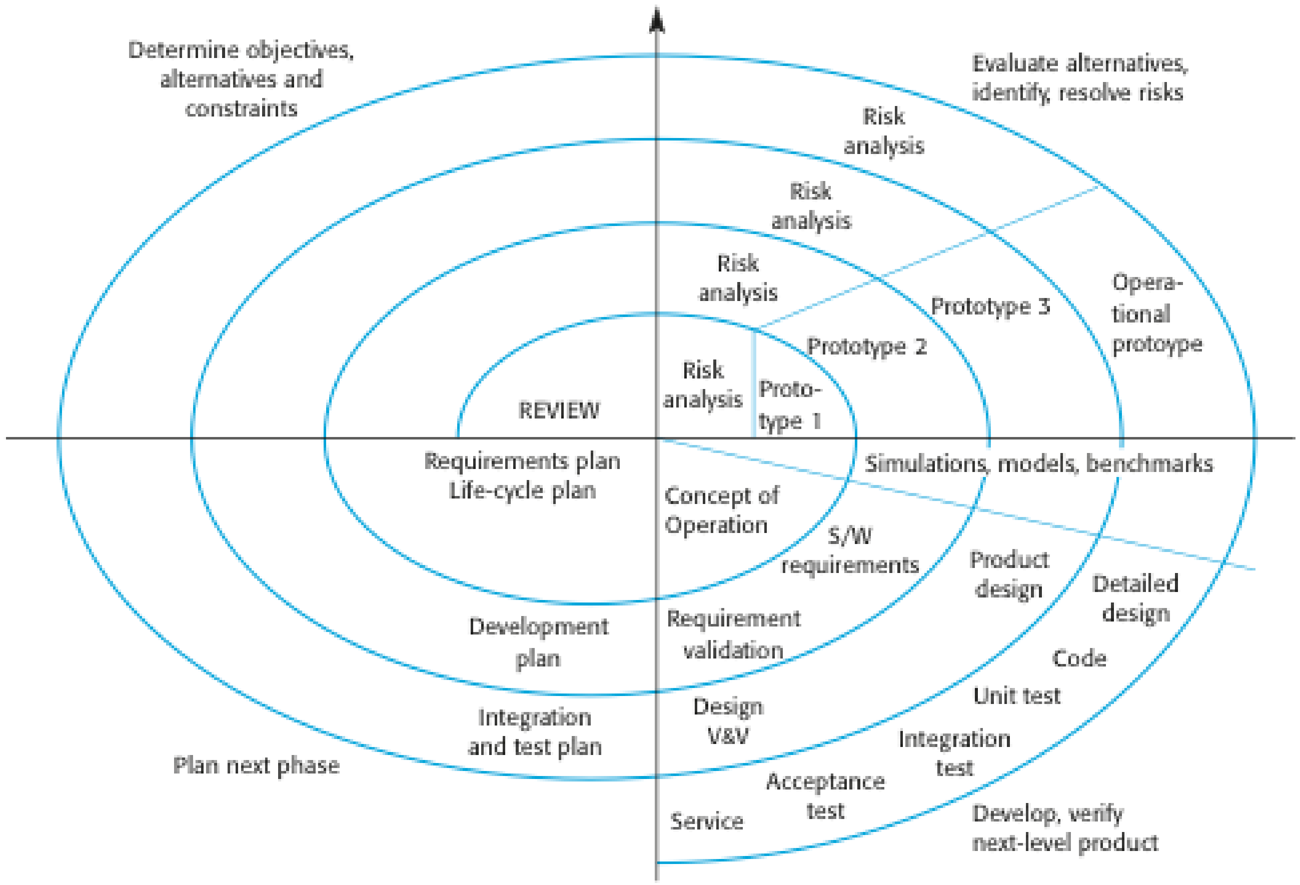
Boehm’s spiral model



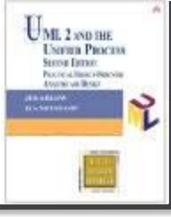
- **Process is represented as a spiral** rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- **No fixed phases** such as specification or design - loops in the spiral are chosen depending on what is required.
- **Risks** are explicitly assessed and resolved throughout the process.

Boehm's spiral model of the software process



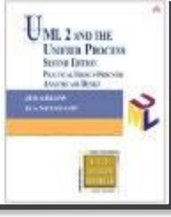


Spiral model sectors

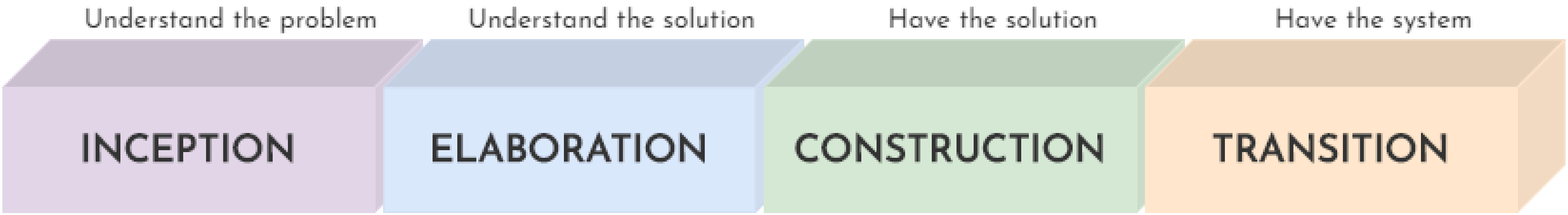


- ✧ Objective setting
  - Specific objectives for the phase are identified.
- ✧ Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks.
- ✧ Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- ✧ Planning
  - The project is reviewed and the next phase of the spiral is planned.

Rational Unified Process (RUP)



- ✧ A modern generic process commonly associated with the Unified Modeling Language (UML).
- ✧ Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time
  - A static perspective that shows process activities
  - A practice perspective that suggests good practices to be used during the process.



Understand the problem

INCEPTION

Understand the solution

ELABORATION

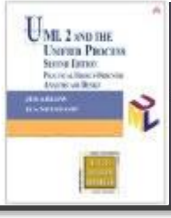
Have the solution

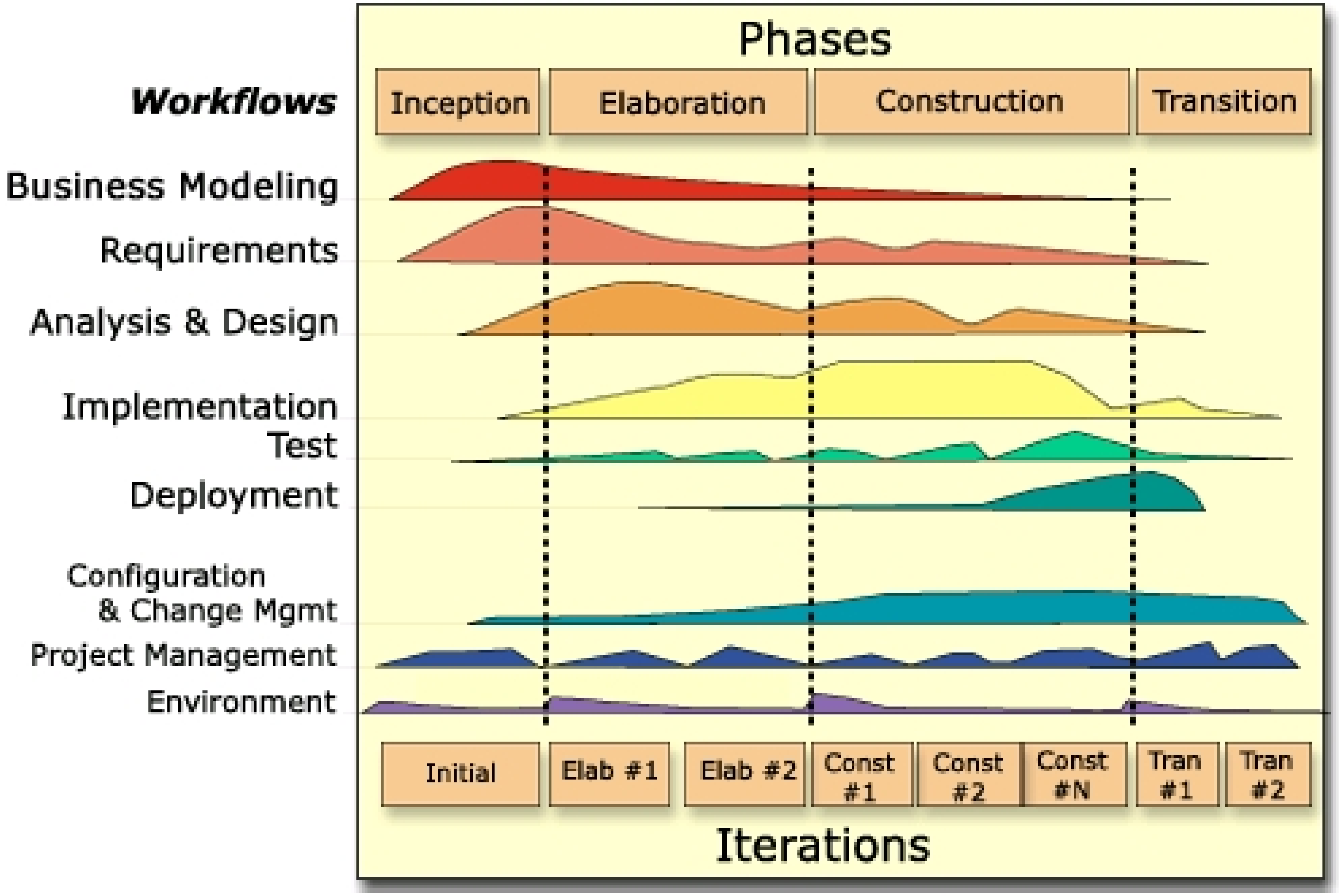
CONSTRUCTION

Have the system

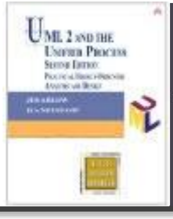
TRANSITION

Rational Unified Process (RUP)

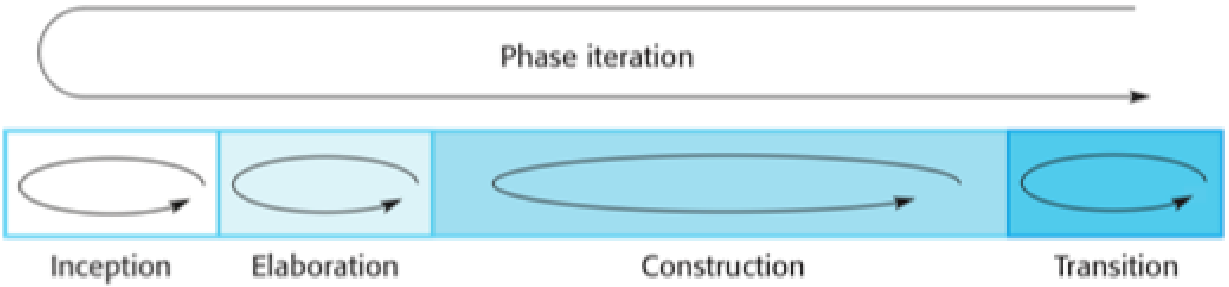




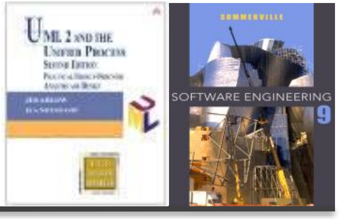
Phases in the Rational Unified Process



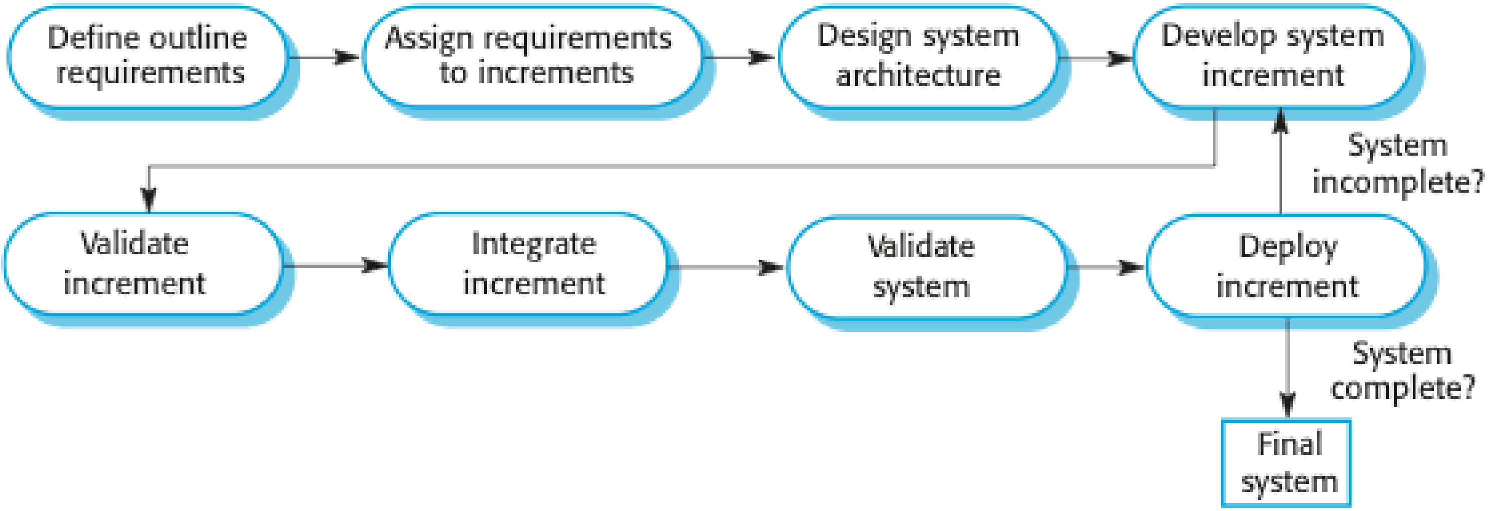
- Inception
  - Establish the business case for the system.
- Elaboration
  - Develop understanding of the problem domain and system architecture.
- Construction
  - System design, programming and testing.
- Transition
  - Deploy the system in its operating environment.



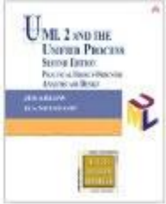
Iterative and incremental development



- ✧ What is the difference between the two?

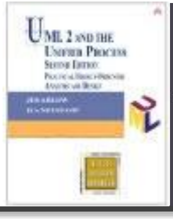


Incremental delivery



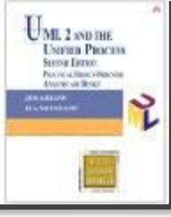
- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with **each increment delivering part of the required functionality**.
- User requirements are **prioritised** and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental development benefits



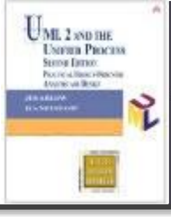
- **Customer value** can be delivered with each increment so system functionality is available earlier.
- Early increments act as a **prototype** to help elicit requirements for later increments.
- **Lower risk** of overall project failure.
- The **highest priority** system services tend to receive the most attention (design, testing, etc.).

Incremental development problems



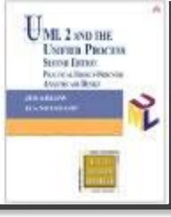
- The **complete specification** is hard to foresee.
  - This becomes problematic when complete specification is required in contract negotiation.
- System **structure tends to degrade** as new increments are added.
  - Unless time and money is spent on extensive **refactoring**, regular changes tend to **corrupt system structure** and increase the cost of incorporating further changes.
- It is hard to identify and effectively design basic **facilities shared** by different parts of the system.
- The process is not visible, **progress is hard to trace**.

Agile methods



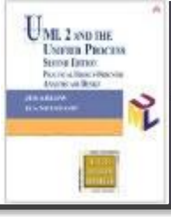
- ✧ Agile methods:
  - Focus on the **code** rather than the design
  - Are based on an **iterative and incremental approach** to software development
  - Are intended to deliver working software quickly and evolve this quickly to **meet changing requirements**.
- ✧ The aim of agile methods is to **reduce overheads in the software process** (e.g. by limiting documentation) and to be able to **respond quickly to changing requirements** without excessive rework.

Reuse-oriented software engineering

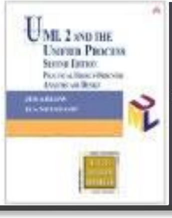


- Based on **systematic reuse** where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.
- Reuse is now the standard approach for building many types of business system

Key points



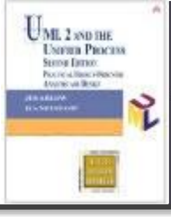
- General process models describe the organization of software processes.
  - Examples of general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.
- Processes should include activities to cope with change.
  - This may involve **prototyping** and **incremental delivery**, which help to avoid poor early decisions on requirements and design.
- Agile methods are incremental development methods that focus on frequent releases, reducing process overheads and emphasize customer involvement.



Agile Development

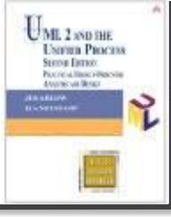
Lecture 4/Part 2

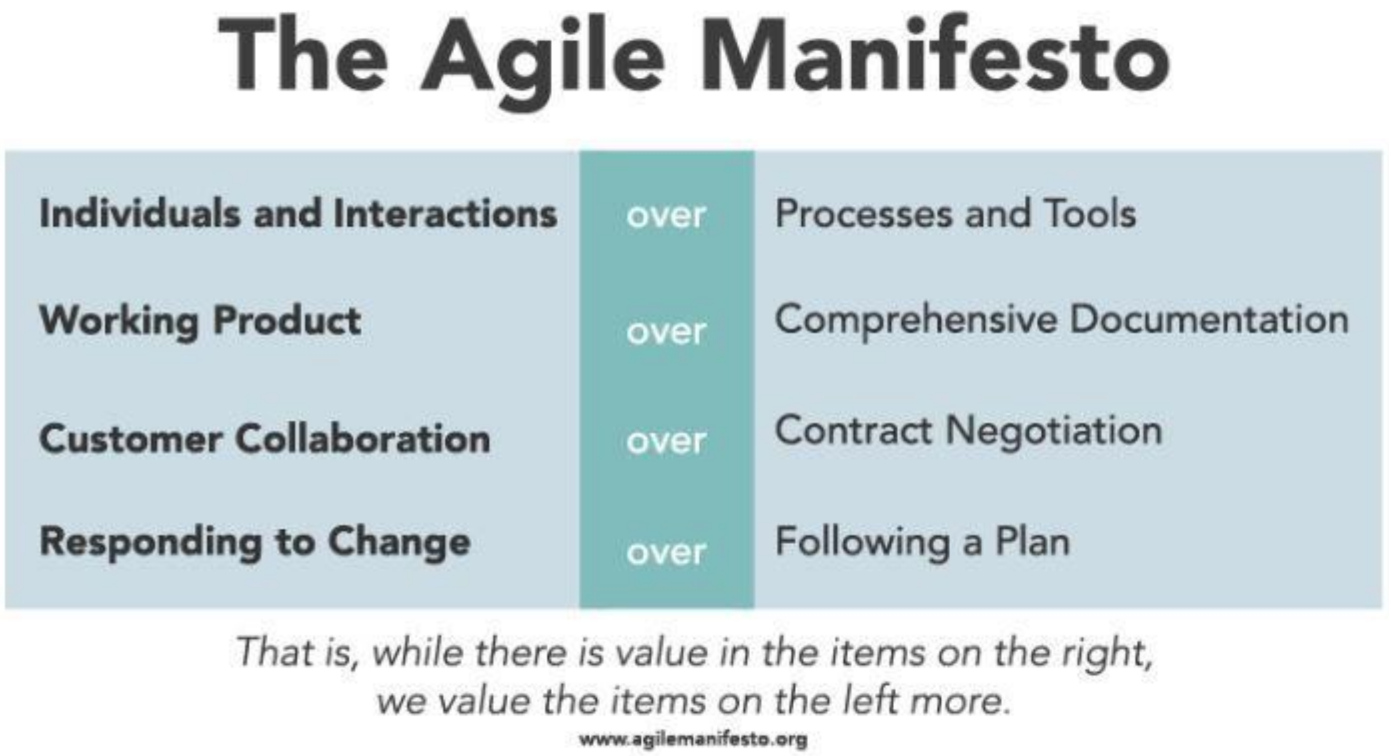
Agile



- ✧ Being agile means being **responsive to a change**
- ✧ A **mindset** established through 4 values, grounded by 12 principles and manifested through many different practices
- ✧ A leadership philosophy that encourages **teamwork**, **self-organization** and **accountability**
- ✧ Main aspects:
  - Flexibility
  - Work breakdown
  - Value of teamwork
  - Iterative improvements
  - Cooperation with a client

Agile manifesto





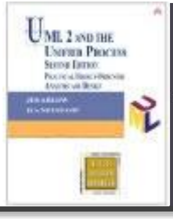
The Agile Manifesto

Individuals and Interactions	over	Processes and Tools
Working Product	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.*

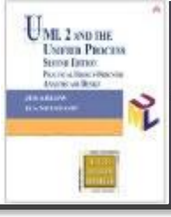
www.agilemanifesto.org

The principles of agile methods

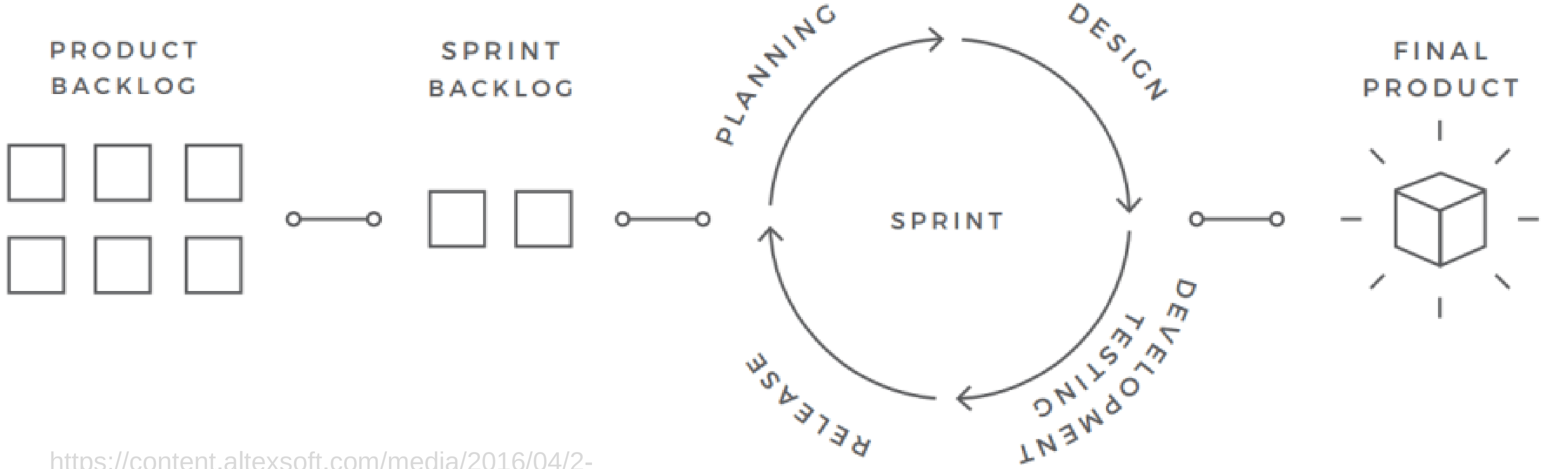


Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Agile development

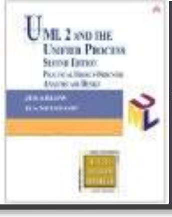


- ✧ A time boxed, **iterative** approach to software delivery that builds software **incrementally** from the start of the project
- ✧ A group of software development methodologies based on iterative development that focuses on frequent releases, reducing process overheads and emphasize customer involvement through collaboration between self-organizing cross-functional teams



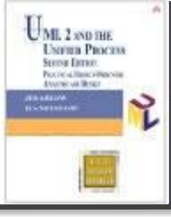
https://content.altexsoft.com/media/2016/04/2-agile-wp-1024x484.png

Benefits of agile development



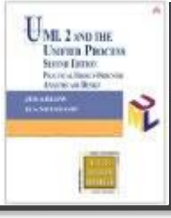
- **Customer satisfaction** by continuous delivery of software
- Working software is **delivered frequently**
- Greater **flexibility** and **adaptability** to change
- Increased **collaboration** frequency and feedback
- Close **cooperation** between stakeholders and developers
- Focused on **Business Value**
- Increased project **control**

Problems and challenges in agile



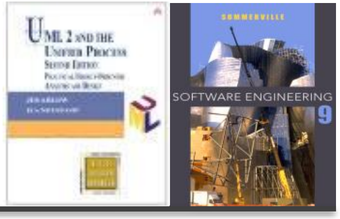
- The project can easily get taken off track if the stakeholder is **not clear** with what final **outcome** they want
- It can be difficult to **keep the interest of customers** who are involved in the process
- The level of **collaboration** can be **difficult** to maintain
- The risk of **losing** long-term **vision** as there is no clear end of the project
- **Contracts** may be a problem as with other approaches to iterative development.

Problems and challenges in agile

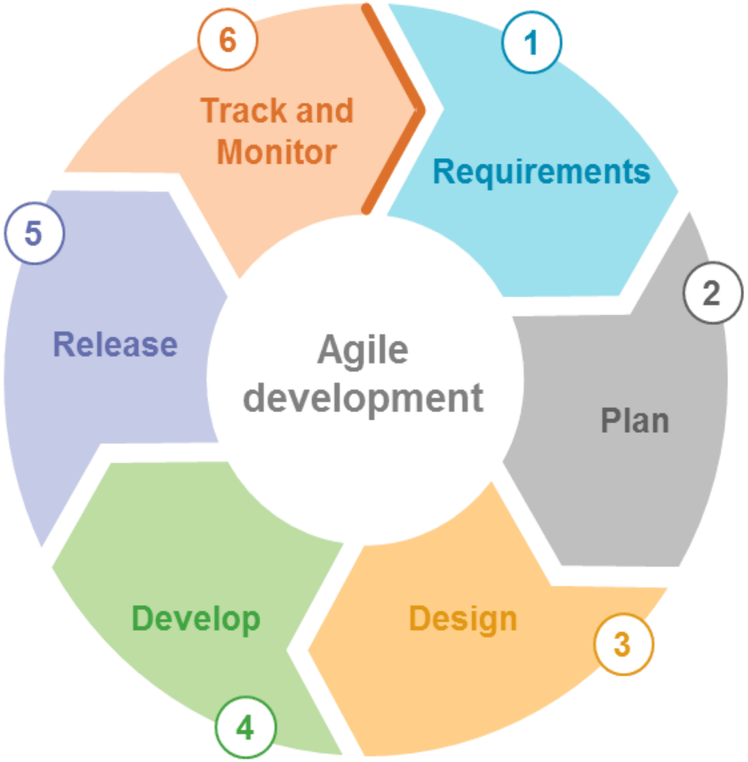


- **Documentation** tends to get **sidetracked**
- **Difficult** to measure **progress**
- Because of their focus on small, tightly-integrated teams, one needs to be careful when **scaling agile methods** to large systems.
- **Prioritizing changes** can be difficult where there are **multiple stakeholders**.
- Maintaining **simplicity** requires extra work

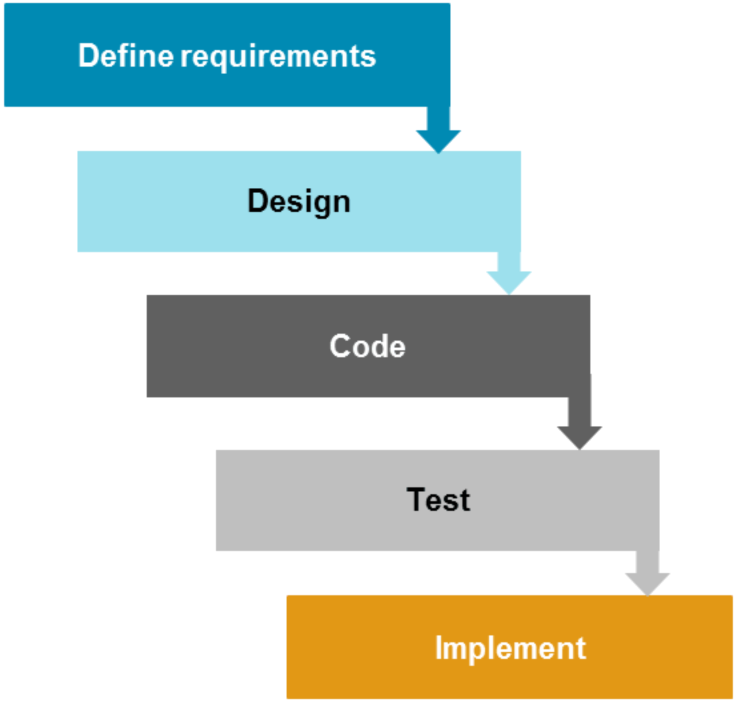
Agile vs Waterfall



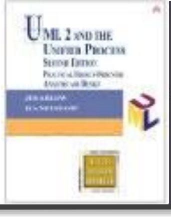
Agile



Waterfall

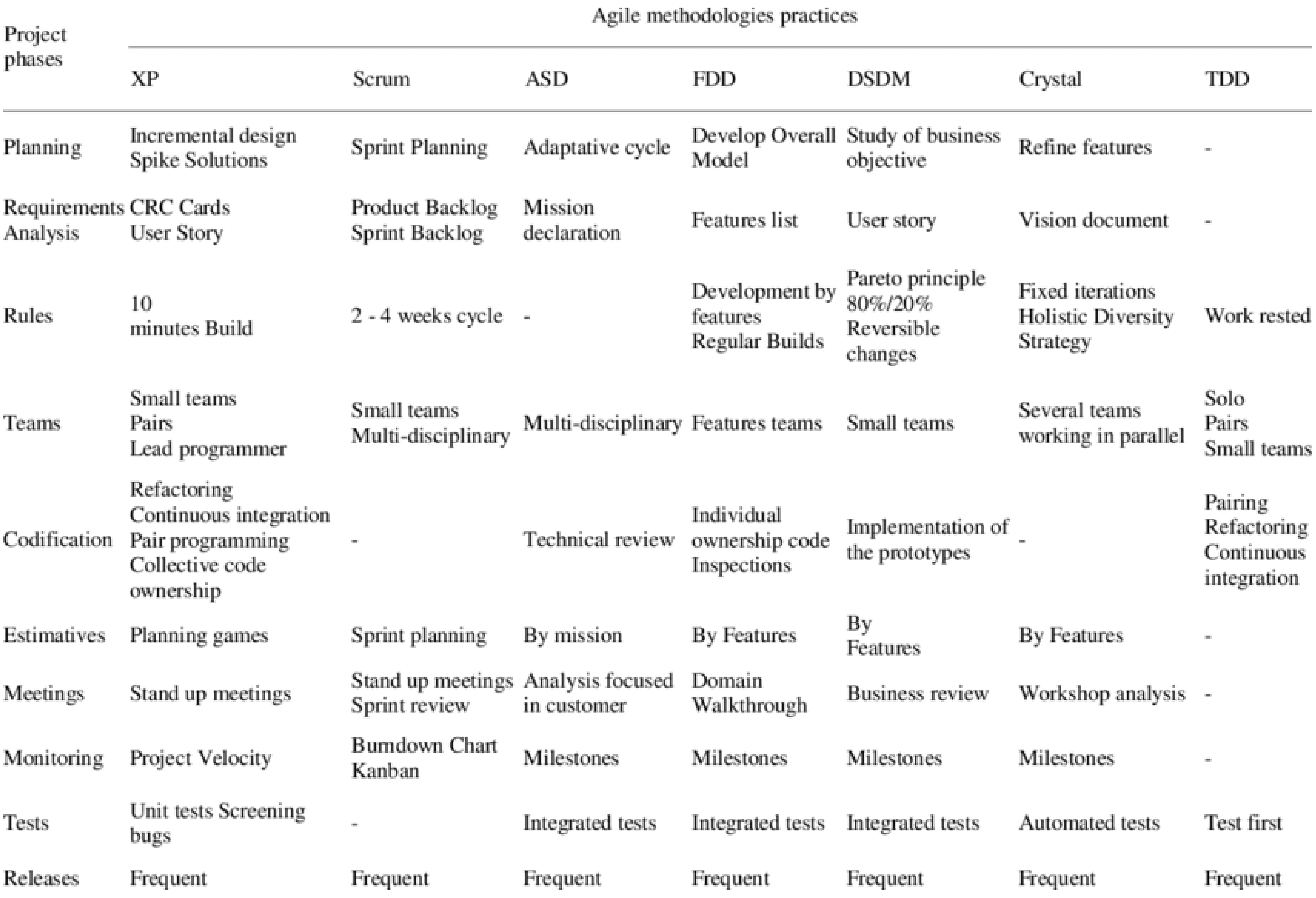


https://saigontechnology.com/assets/media/agile-scrum-vs-waterfall.png



Agile Practices

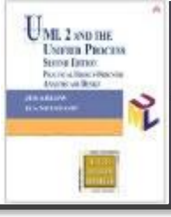
Lecture 4/Part 3



Project phases	Agile methodologies practices						
	XP	Scrum	ASD	FDD	DSDM	Crystal	TDD
Planning	Incremental design Spike Solutions	Sprint Planning	Adaptative cycle	Develop Overall Model	Study of business objective	Refine features	-
Requirements Analysis	CRC Cards User Story	Product Backlog Sprint Backlog	Mission declaration	Features list	User story	Vision document	-
Rules	10 minutes Build	2 - 4 weeks cycle	-	Development by features Regular Builds	Pareto principle 80%/20% Reversible changes	Fixed iterations Holistic Diversity Strategy	Work rested
Teams	Small teams Pairs Lead programmer	Small teams Multi-disciplinary	Multi-disciplinary	Features teams	Small teams	Several teams working in parallel	Solo Pairs Small teams
Codification	Refactoring Continuous integration Pair programming Collective code ownership	-	Technical review	Individual ownership code Inspections	Implementation of the prototypes	-	Pairing Refactoring Continuous integration
Estimatives	Planning games	Sprint planning	By mission	By Features	By Features	By Features	-
Meetings	Stand up meetings	Stand up meetings Sprint review	Analysis focused in customer	Domain Walkthrough	Business review	Workshop analysis	-
Monitoring	Project Velocity	Burndown Chart Kanban	Milestones	Milestones	Milestones	Milestones	-
Tests	Unit tests Screening bugs	-	Integrated tests	Integrated tests	Integrated tests	Automated tests	Test first
Releases	Frequent	Frequent	Frequent	Frequent	Frequent	Frequent	Frequent

https://www.researchgate.net/publication/267429278_Agile_Practices_An_Assessment_of_Perception_of_Value_of_Professionals_on_the_Quality_Criteria_in_Performance_of_Projects

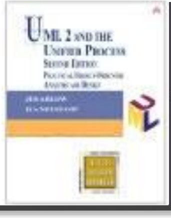
User Story



- The smallest unit of work in an agile framework
- An informal, natural language description of a feature or desired outcome of a software system
- Often written from the perspective of an end user of a system to influence the functionality of the system being developed
- May be written by various stakeholders including clients, users, managers, or development team members

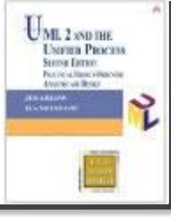
As a < type of user >, I want < some goal >
so that < some reason >.

Daily Scrum (Stand-up)

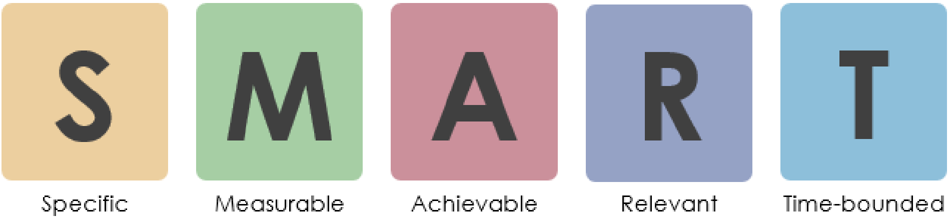


- A 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours
- Optimizes team collaboration and performance by sharing the work done since the last Daily Scrum/Stand-up and forecasting upcoming Sprint work
- The Daily Scrum/Stand-up is held at the same time and place each day
- Every team member should answer these questions:
  - *What did I work on yesterday?*
  - *What am I working on today?*
  - *What issues are blocking me?*

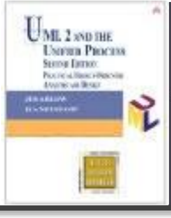
Backlog



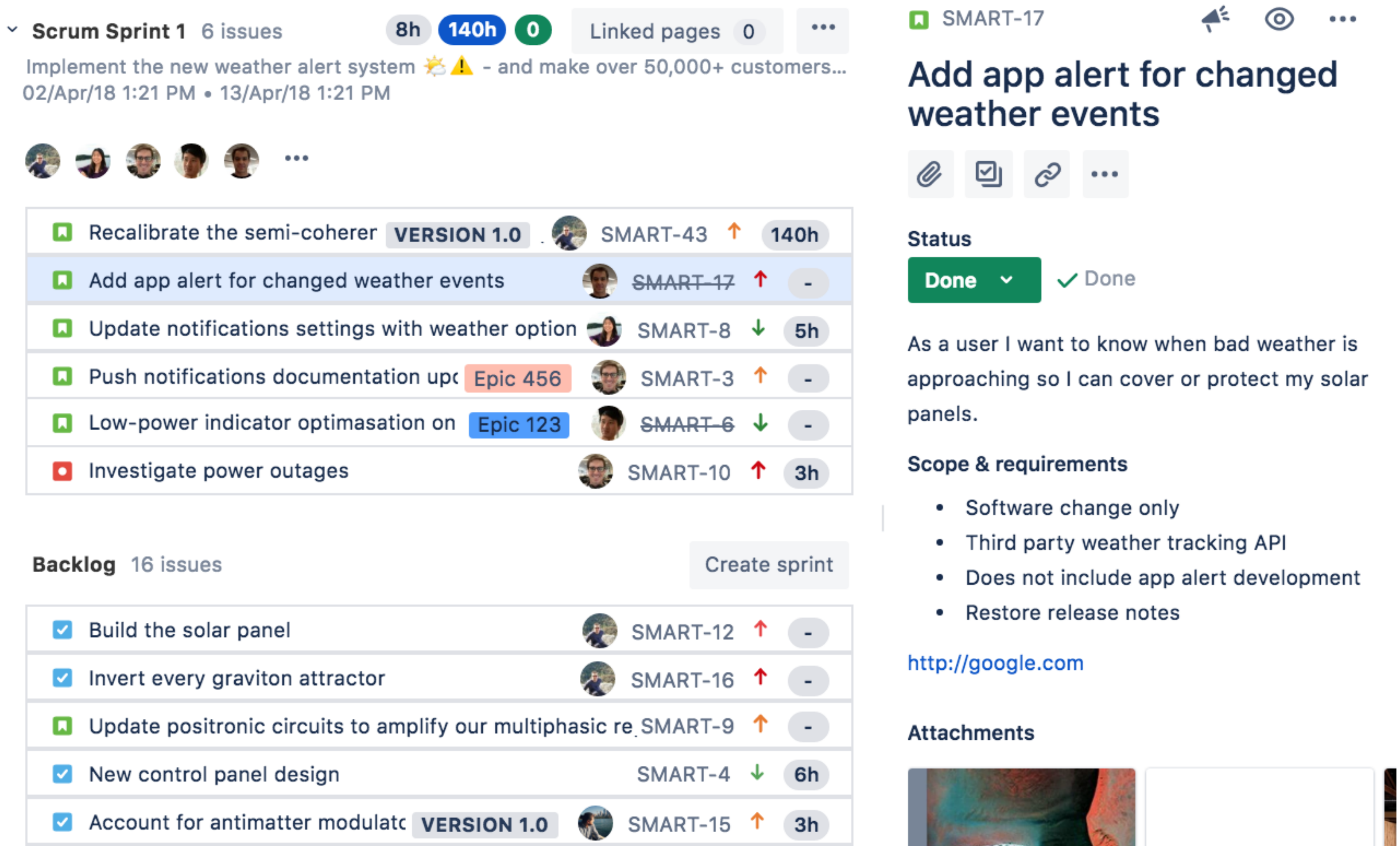
- ✧ List of items ordered by priority, prioritized by the product owner
- ✧ The items ranked highest on the list represent the most important or urgent items for the team to complete
- ✧ **Product backlog**:
  - The list of tasks to be done and contains a prioritized list of all product requirements that a team maintains for a product
- ✧ **Sprint backlog**:
  - The list of tasks from product backlog to be completed by the development team during the next sprint



Backlog



Sprint backlog



Scrum Sprint 1 6 issues 8h 140h 0 Linked pages 0 •••

Implement the new weather alert system 🌤⚠ - and make over 50,000+ customers...

02/Apr/18 1:21 PM • 13/Apr/18 1:21 PM

🟩	Recalibrate the semi-coherer VERSION 1.0	SMART-43	↑	140h
🟩	Add app alert for changed weather events	~~SMART-17~~	↑	-
🟩	Update notifications settings with weather option	SMART-8	↓	5h
🟩	Push notifications documentation upc Epic 456	SMART-3	↑	-
🟩	Low-power indicator optimasation on Epic 123	~~SMART-6~~	↓	-
🟥	Investigate power outages	SMART-10	↑	3h

Backlog 16 issues Create sprint

☑	Build the solar panel	SMART-12	↑	-
☑	Invert every graviton attractor	SMART-16	↑	-
🟩	Update positronic circuits to amplify our multiphasic re	SMART-9	↑	-
☑	New control panel design	SMART-4	↓	6h
☑	Account for antimatter modulatc VERSION 1.0	SMART-15	↑	3h

🟩 SMART-17

Add app alert for changed weather events

Status

Done ⌄ ✓ Done

As a user I want to know when bad weather is approaching so I can cover or protect my solar panels.

Scope & requirements

- Software change only
- Third party weather tracking API
- Does not include app alert development
- Restore release notes

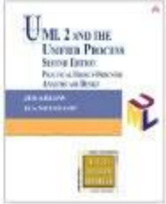
http://google.com

Attachments

User Story

Product backlog

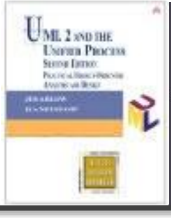
© Papcunová

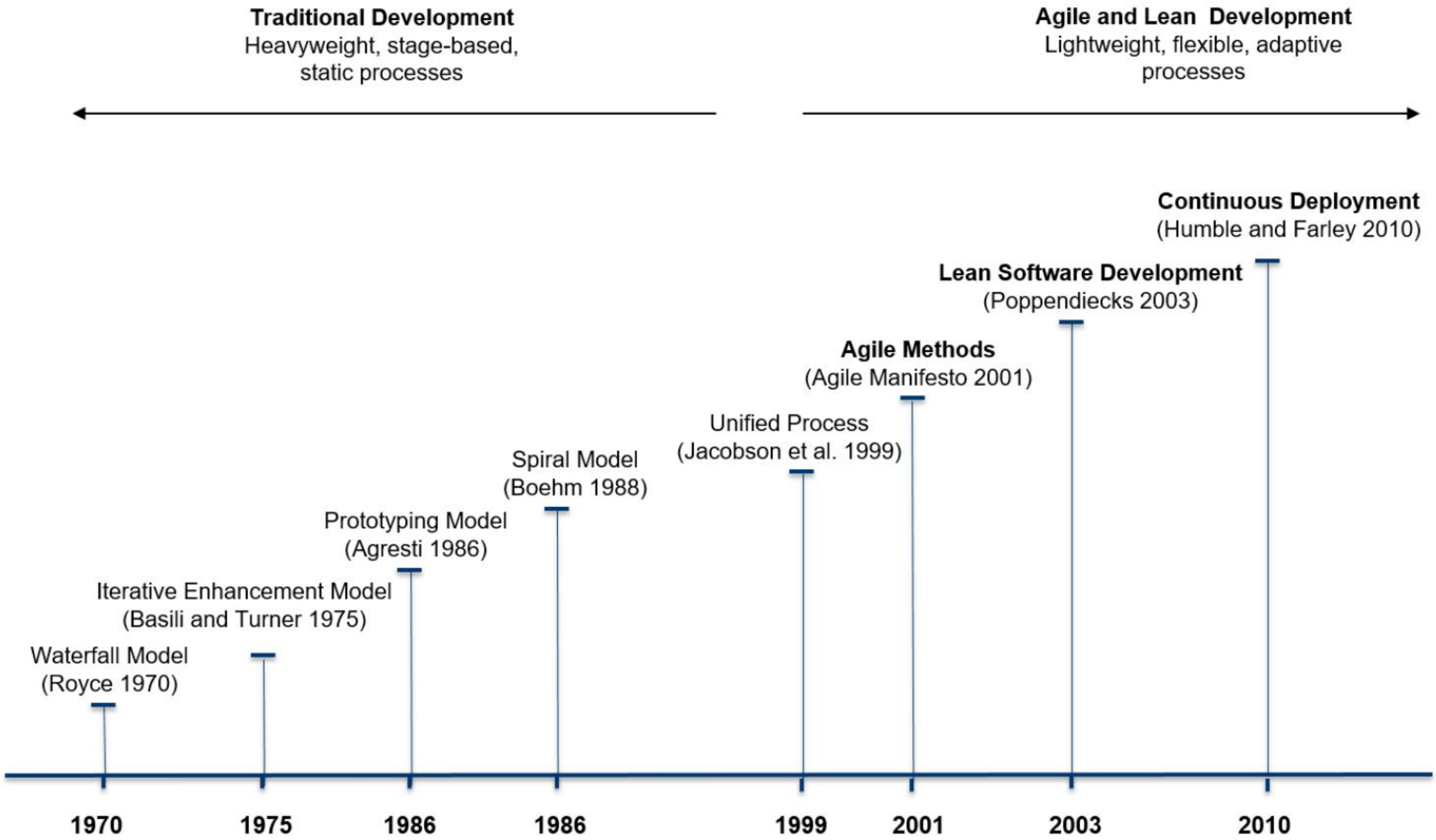


Agile Methods

Lecture 4/Part 4

History of agile





Traditional Development
Heavyweight, stage-based,
static processes

Agile and Lean Development
Lightweight, flexible, adaptive
processes

Waterfall Model
(Royce 1970)

Iterative Enhancement Model
(Basili and Turner 1975)

Prototyping Model
(Agresti 1986)

Spiral Model
(Boehm 1988)

Unified Process
(Jacobson et al. 1999)

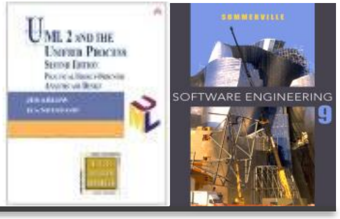
Agile Methods
(Agile Manifesto 2001)

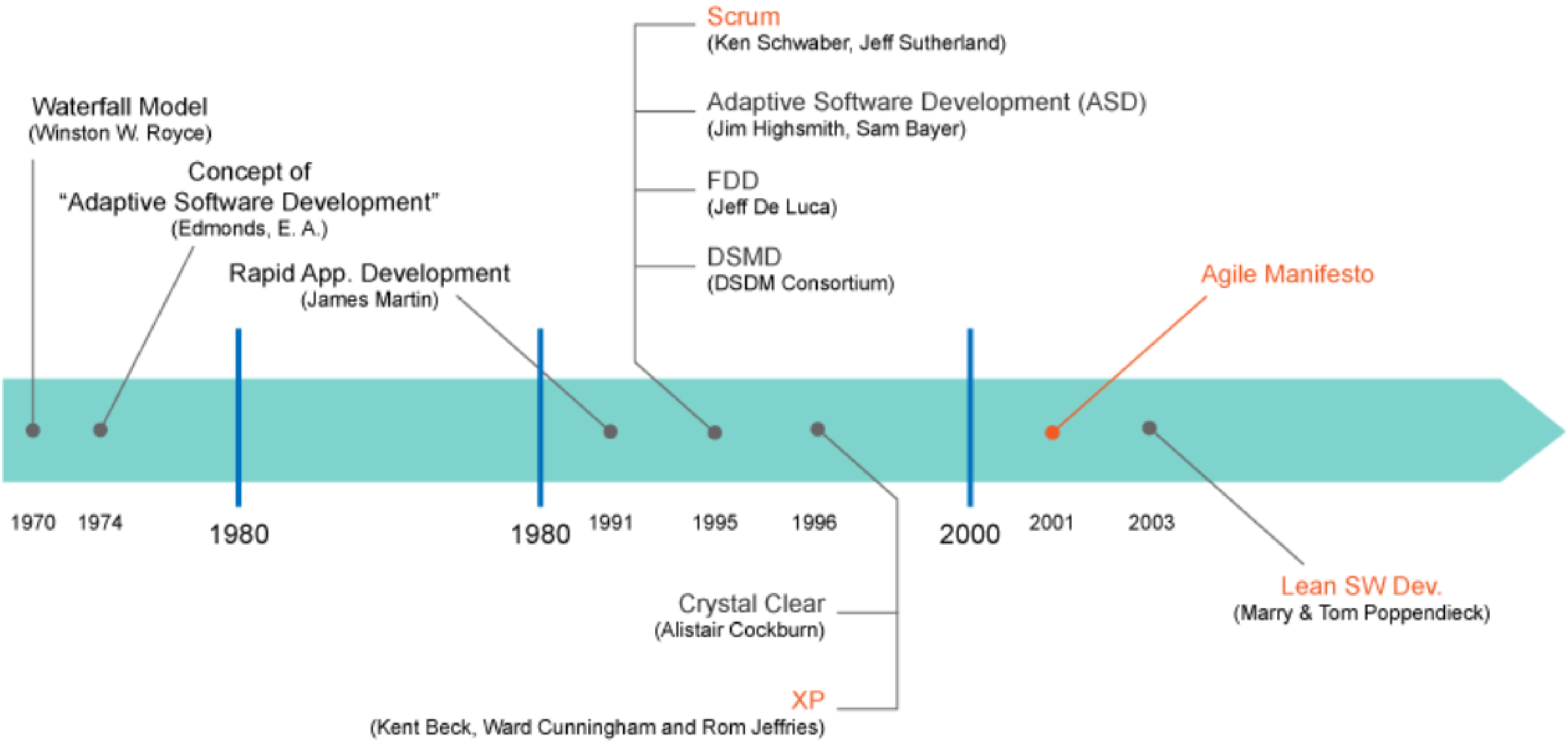
Lean Software Development
(Poppendiecks 2003)

Continuous Deployment
(Humble and Farley 2010)

Rodríguez, Pilar, Mika Mäntylä, Markku Oivo, Lucy Ellen Lwakatare, Pertti Seppänen, and Pasi Kuvaja. "Advances in using agile and lean processes for software development." In Advances in Computers, vol. 113, pp. 135-224. Elsevier, 2019.

History of agile




Waterfall Model
(Winston W. Royce)

Concept of
"Adaptive Software Development"
(Edmonds, E. A.)

Rapid App. Development
(James Martin)

Scrum
(Ken Schwaber, Jeff Sutherland)

Adaptive Software Development (ASD)
(Jim Highsmith, Sam Bayer)

FDD
(Jeff De Luca)

DSMD
(DSDM Consortium)

Agile Manifesto

1970 1974 1980 1980 1991 1995 1996 2000 2001 2003

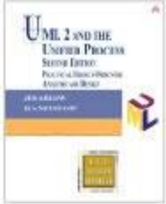
Crystal Clear
(Alistair Cockburn)

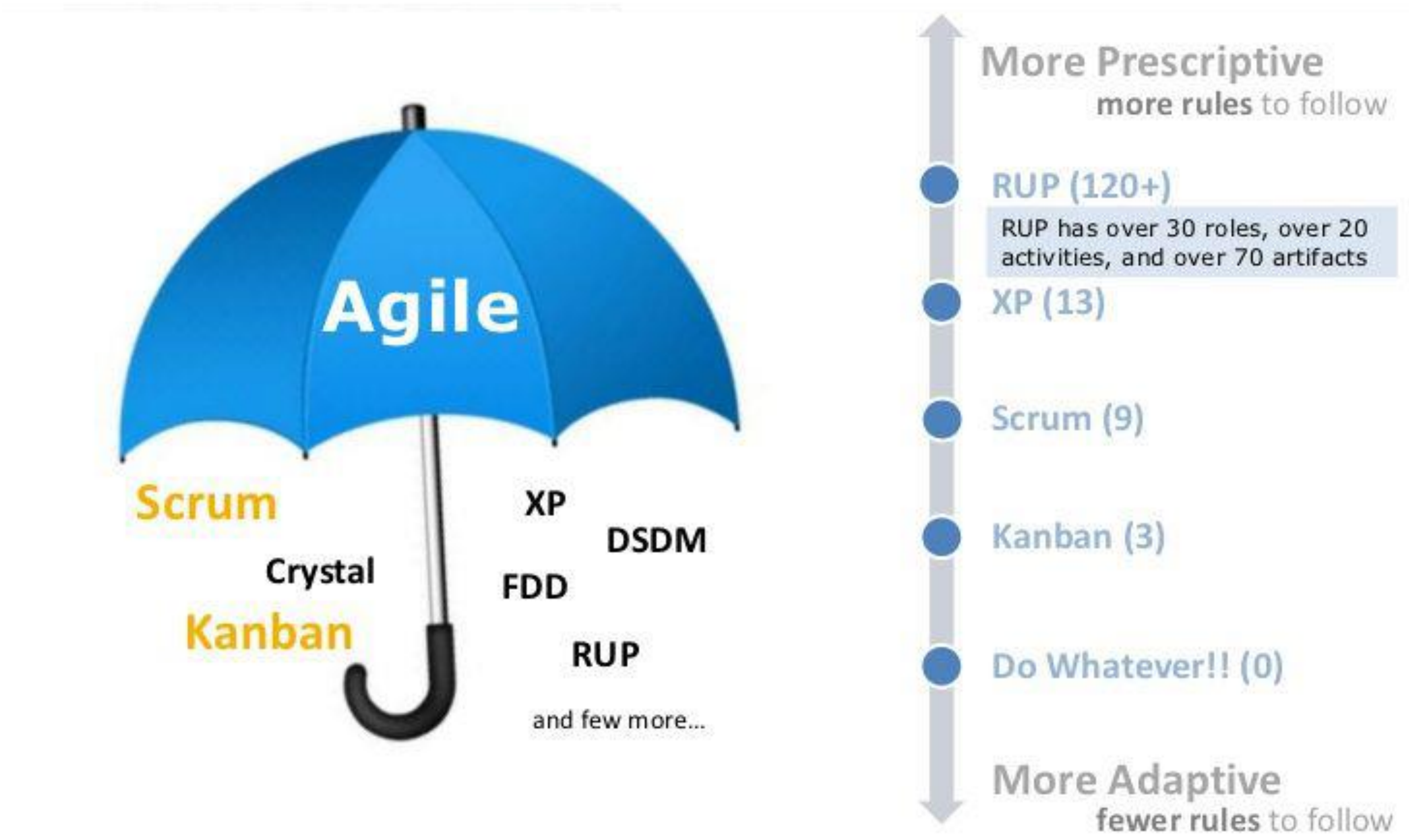
XP
(Kent Beck, Ward Cunningham and Rom Jeffries)

Lean SW Dev.
(Marry & Tom Poppendieck)

https://www.visual-paradigm.com/guide/agile-software-development/what-is-agile-software-development/

Agile umbrella

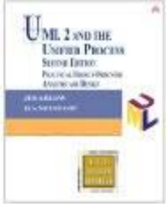




RUP has over 30 roles, over 20 activities, and over 70 artifacts

https://i.pinimg.com/originals/e8/8f/1d/e88f1dfcf8917879c7391f42eef449fa.jpg

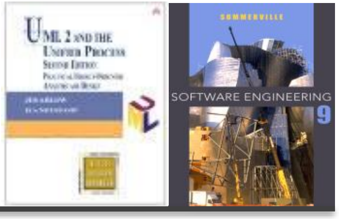
Common points



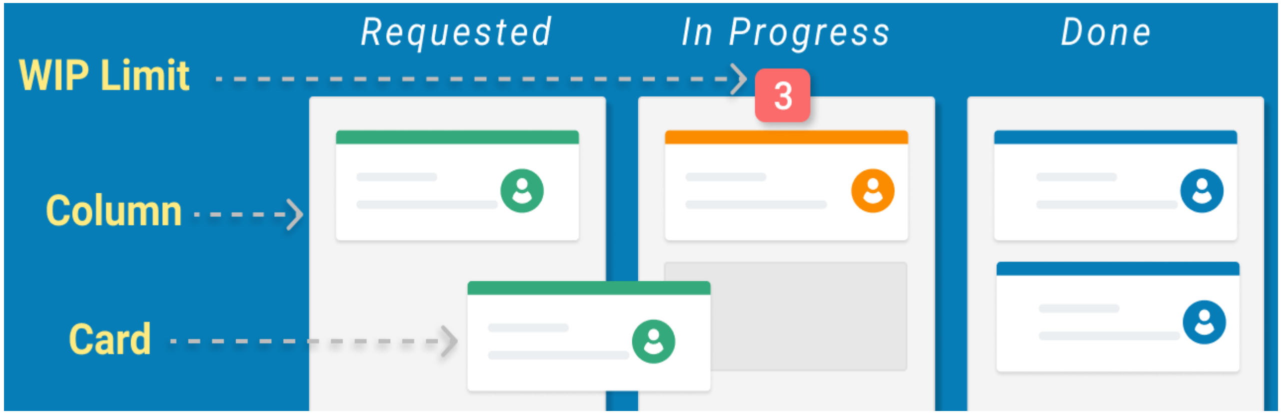
✧ All Agile methods have these points in common:

- **Iterative design process**
- **Effective communication and stakeholder engagement**
- **Aiming for quality and reliable software**
- **Short development cycle allowing regular delivery of software**

Kanban

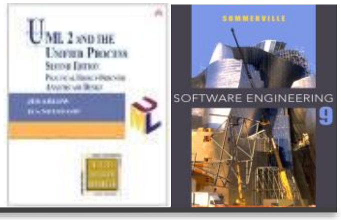


- ✧ A workflow designed to help visualize the work, maximize efficiency requiring real-time communication of capacity and full transparency of work
- ✧ Work items are represented visually on a kanban board, allowing team members to see the state of every piece of work at any time
- ✧ Two main practices are:
  - Visualize your work
  - Limit work in progress (WIP)

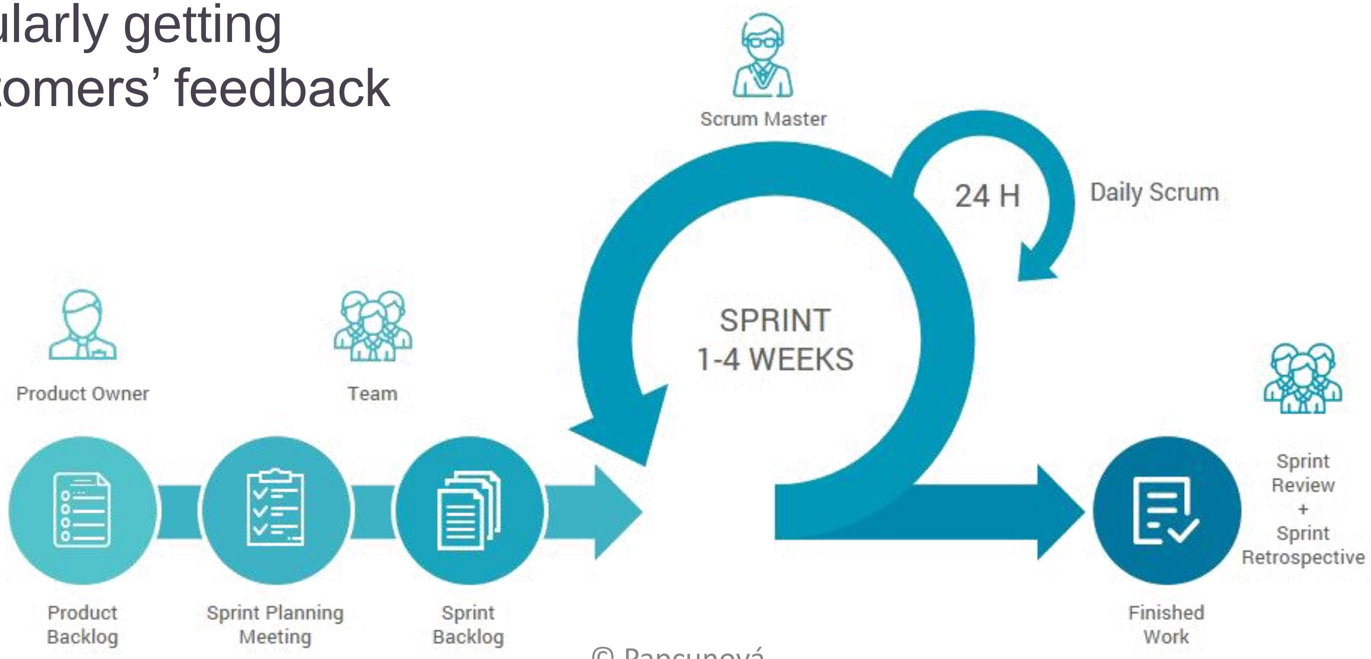


https://kanbanize.com/wp-content/uploads/website-images/kanban-resources/Kanban_board_elements.png

Scrum

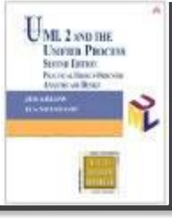


- A set of meetings, tools, practices and roles to help teams structure and manage their work
- Teams deliver products in iterations called sprints
- Continuously creating the highest priority parts of functionality and regularly getting customers’ feedback



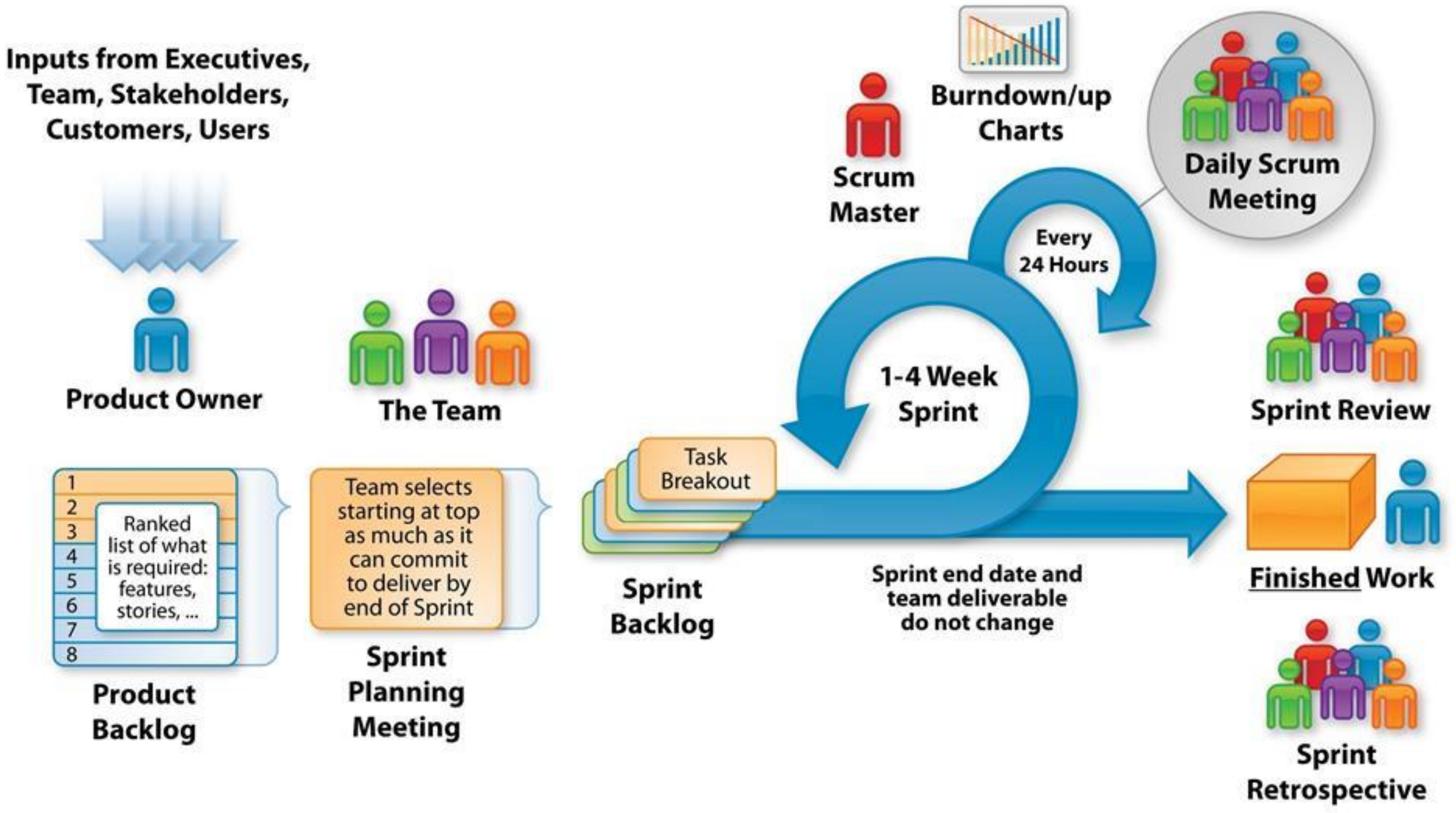
https://media.vlpt.us/images/katanazero86/post/dca8baba-9834-4276-ae85-870753abb0dd/%EB%8B%A4%EC%9A%B4%EB%A1%9C%EB%93%9C.jpg?w=768

Scrum ceremonies



- **Sprint**
  - Basic unit of development in Scrum, fixed duration 1-4 weeks.
- **Sprint planning**
  - Planning event to discuss and agree on the scope of work that is intended to be done during that sprint.
- **Daily Scrum**
  - Each day during a sprint, the team holds a daily scrum (or stand-up) to let everybody say what they completed yesterday, what they plan to complete today, what impediment they face.
- **Sprint review**
  - The team reviews the work that was completed and plans the work that was not completed.
- **Sprint retrospective**
  - Team reflects on the past sprint and identifies and agrees on continuous process improvement actions.

The Agile - Scrum Framework

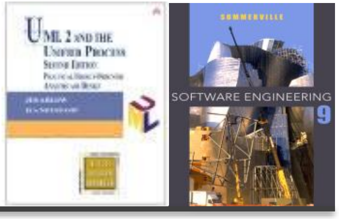


Ranked list of what is required: features, stories, ...

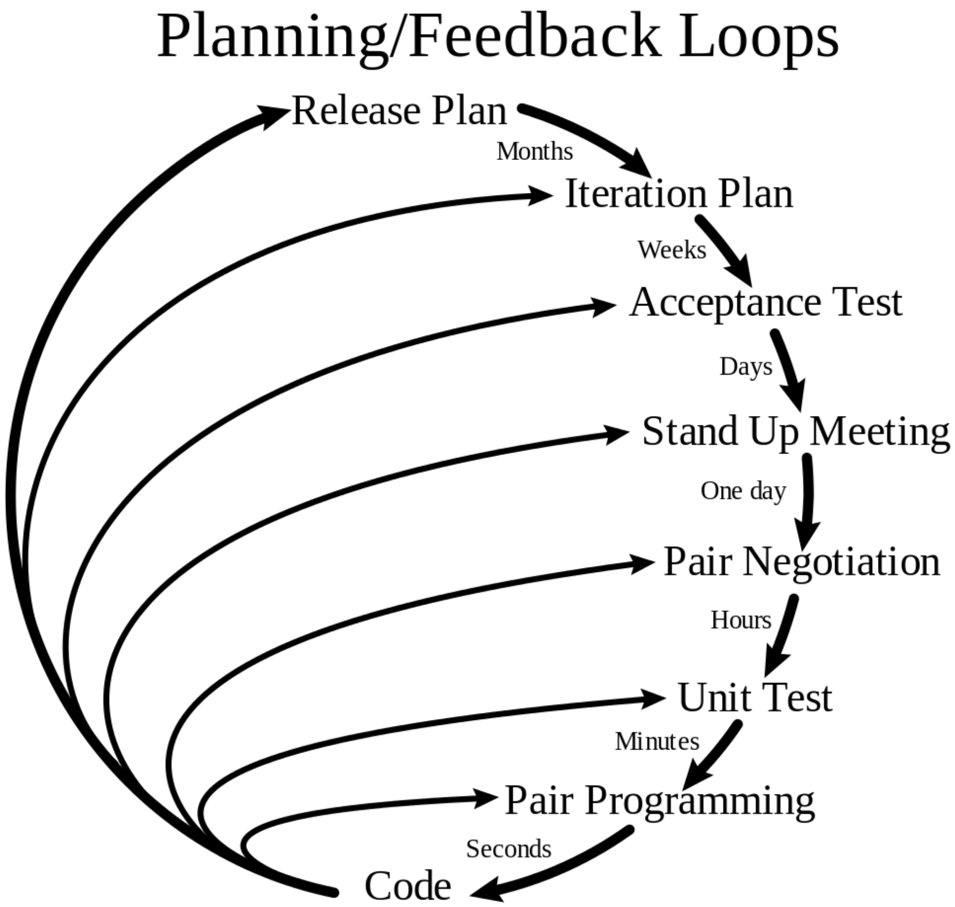
Team selects starting at top as much as it can commit to deliver by end of Sprint

Sprint end date and team deliverable do not change

Extreme programming

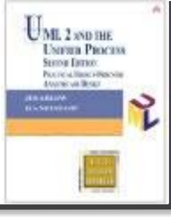


- ✧ An agile methodology designed to improve the quality of software and its ability to adapt to the changing needs of the customer
- ✧ Iterative and frequent small releases
- ✧ Practices:
  - Pair programming
  - Test driven development (TDD)



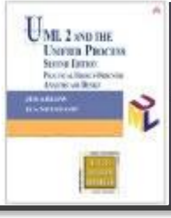
https://www.hiclipart.com/free-transparent-background-png-clipart-aqzrw

Feature driven development

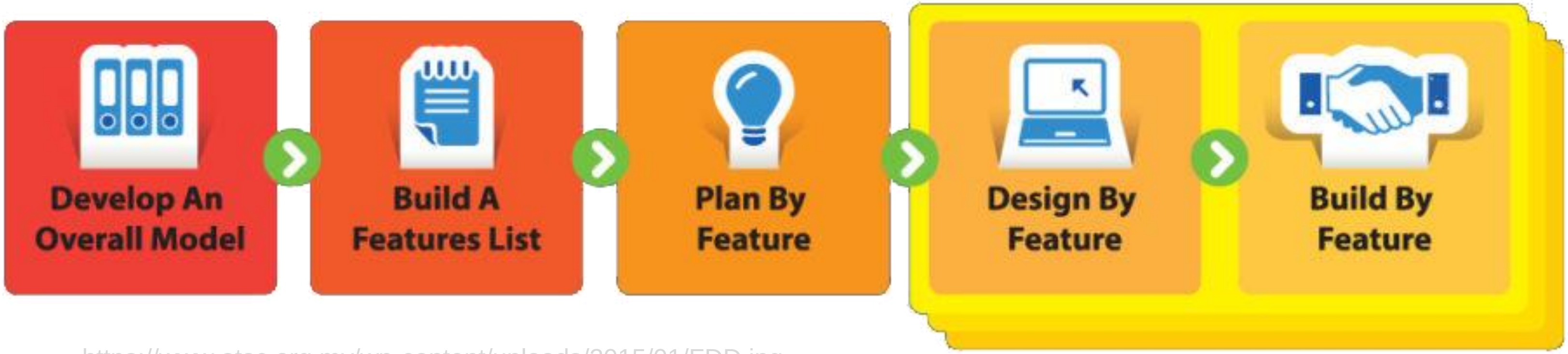


- A client-centered, architecture-centered, and pragmatic software process
- Ideal for long-term, complex projects looking for a simple but comprehensive methodology with clear outcomes
- Principles
  - Domain object modeling
  - Developing by feature
  - Individual class ownership
  - Feature teams
  - Inspections
  - Configuration management
  - Progress reports

Feature driven development



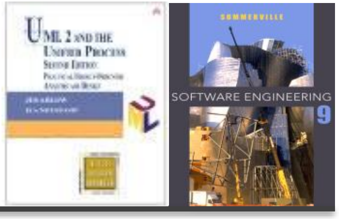
- ✧ Project stages:
  - Develop An Overall Model
  - Build a Features List
  - Plan By Feature
  - Design By Feature
  - Build By Feature



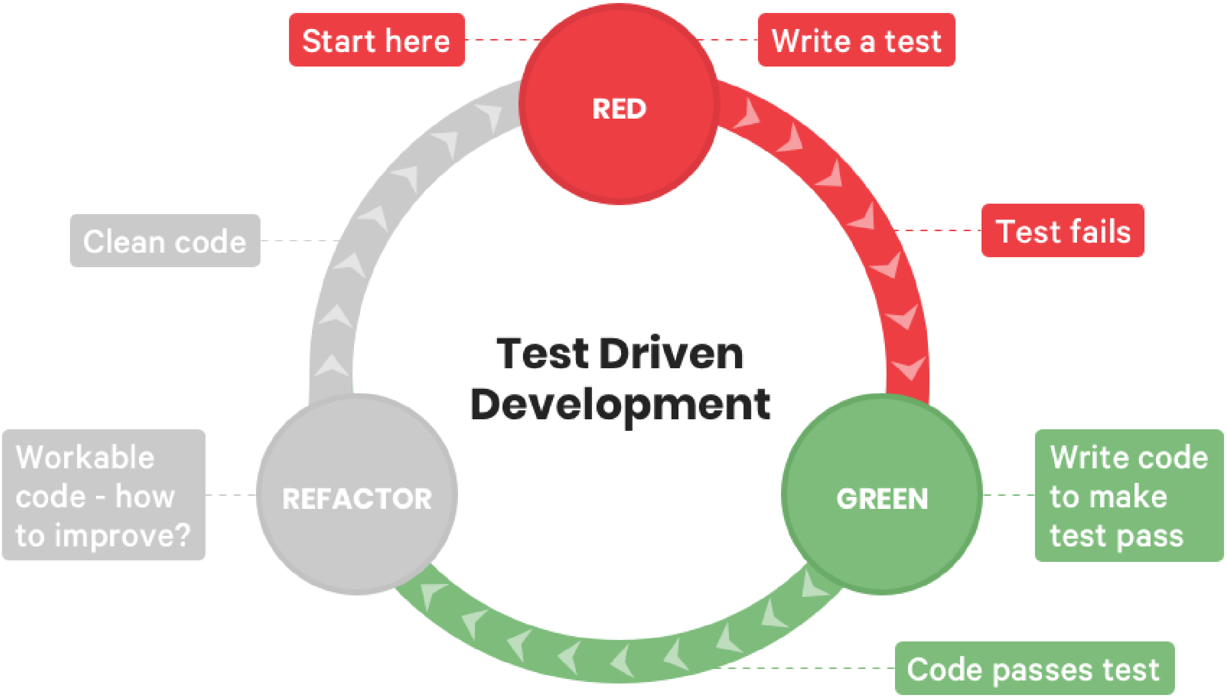
Develop An Overall Model › Build A Features List › Plan By Feature › Design By Feature › Build By Feature

https://www.atsc.org.my/wp-content/uploads/2015/01/FDD.jpg

Test driven development



- ✧ A process that relies on the repetition of a very short development cycle: requirements are turned into very specific unit test cases, then the code is improved so that the tests pass



Start here

Write a test

RED

Test fails

Test Driven Development

GREEN

Write code to make test pass

Code passes test

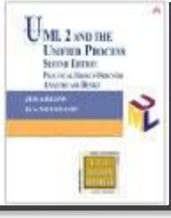
REFACTOR

Workable code - how to improve?

Clean code

https://codica-images-staging.s3.eu-central-1.amazonaws.com/c3cb64e3ac2d4eae89002e0ccc5789dd.png

Project roles



✧ Product Owner

- An expert on the product and the customer’s needs and priorities. Works with the development team daily to help clarify requirements and makes business decisions.

✧ Scrum Master

- The team role responsible for ensuring the team lives agile values and principles and follows the processes and practices that the team agreed they would use.

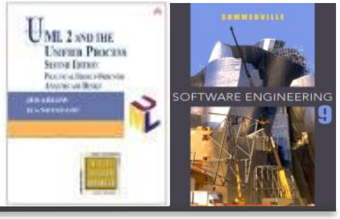
✧ Team Member

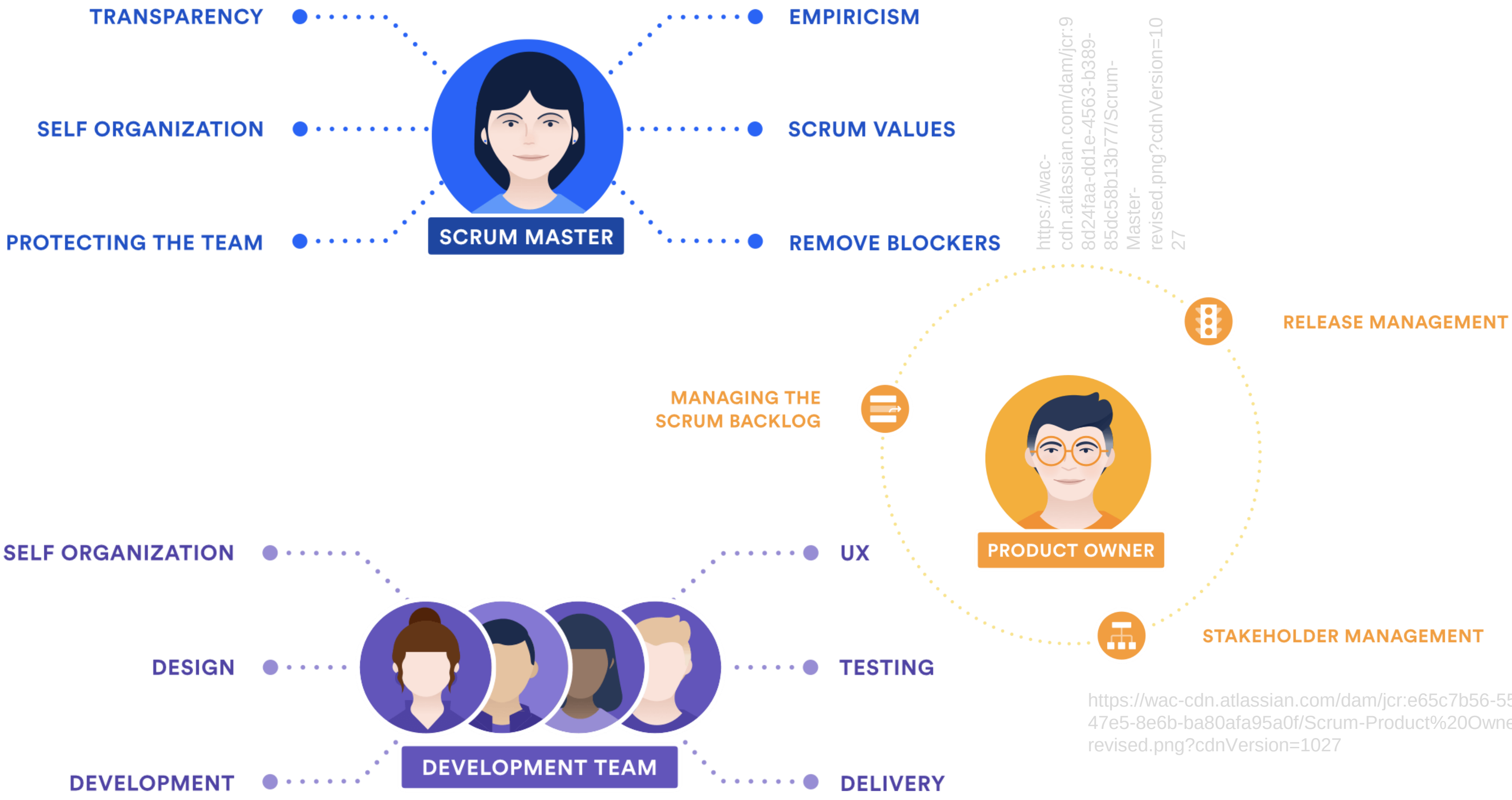
- The people who create the product. Programmers, testers, designers, writers, data engineers, and anyone else with a hands-on role in product development.

✧ Stakeholder

- Anyone with an interest in the project. Provides regular feedback and is affected by the project’s outcome.

Project roles




TRANSPARENCY

SELF ORGANIZATION

PROTECTING THE TEAM

SCRUM MASTER

EMPIRICISM

SCRUM VALUES

REMOVE BLOCKERS

https://wac-cdn.atlassian.com/dam/jcr:98d24faa-dd1e-4563-b389-85dc58b13b77/Scrum-Master-revised.png?cdnVersion=1027

RELEASE MANAGEMENT

MANAGING THE SCRUM BACKLOG

PRODUCT OWNER

STAKEHOLDER MANAGEMENT

SELF ORGANIZATION

DESIGN

DEVELOPMENT

DEVELOPMENT TEAM

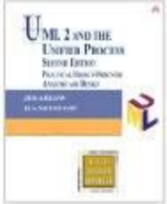
UX

TESTING

DELIVERY

https://wac-cdn.atlassian.com/dam/jcr:e65c7b56-55e3-47e5-8e6b-ba80afa95a0f/Scrum-Product%20Owner-revised.png?cdnVersion=1027

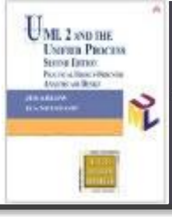
https://wac-cdn.atlassian.com/dam/jcr:f085fea0-5149-4b9a-9fe1-7e9fd32dc0da/Scrum-Development%20team-revised.png?cdnVersion=1027



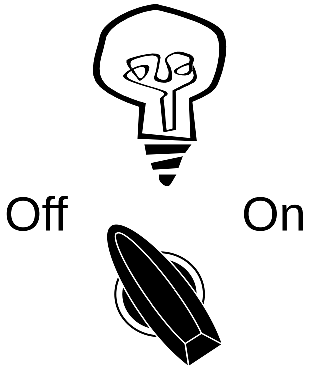
UML State Diagram

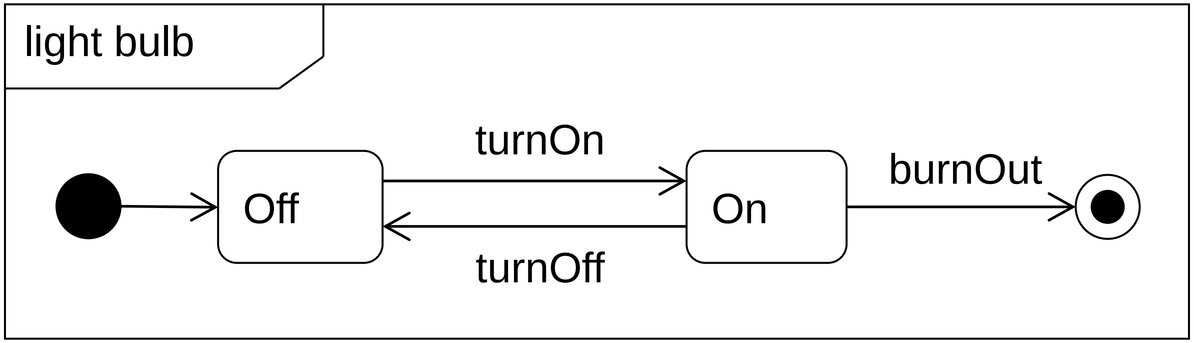
Lecture 4/Part 5

State machines

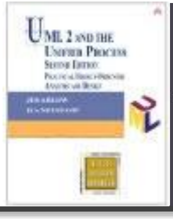


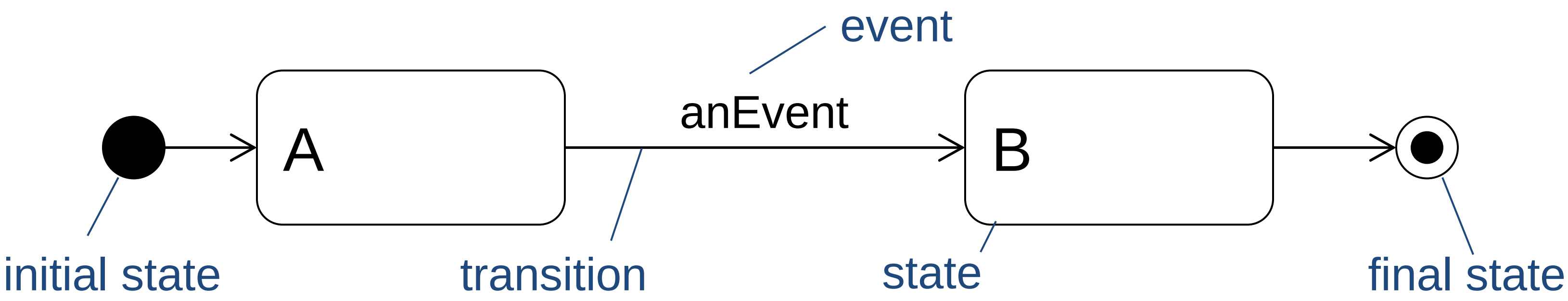
- ✧ Models life stages of a **single** model element – e.g. object, use case, module
- ✧ Every state machine exists in the context of a particular model element that:
  - Has a clear life history modelled as a progression of **states**, **transitions** and **events**
  - Responds to events dispatched from outside of the element
- ✧ There are two types of state machines:
  - **Behavioural state machines** - define the behaviour of a model element
  - **Protocol state machines** - model the protocol of a classifier
    - E.g. call conditions and call ordering of an interface that itself has no behaviour





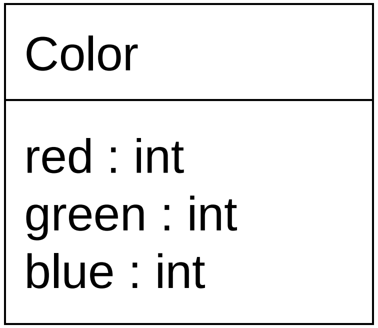
Basic state machine syntax



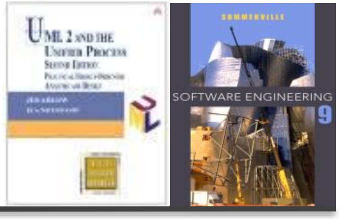


- State = a situation or condition during the life of an object
  - Determined at any point in time by the **values of its attributes**, the relationships to other objects, or the **activities** it is performing.
- Every state machine should have one initial state which indicates the first state of the sequence
- Unless the states cycle endlessly, state machines should have a final state which terminates its lifecycle

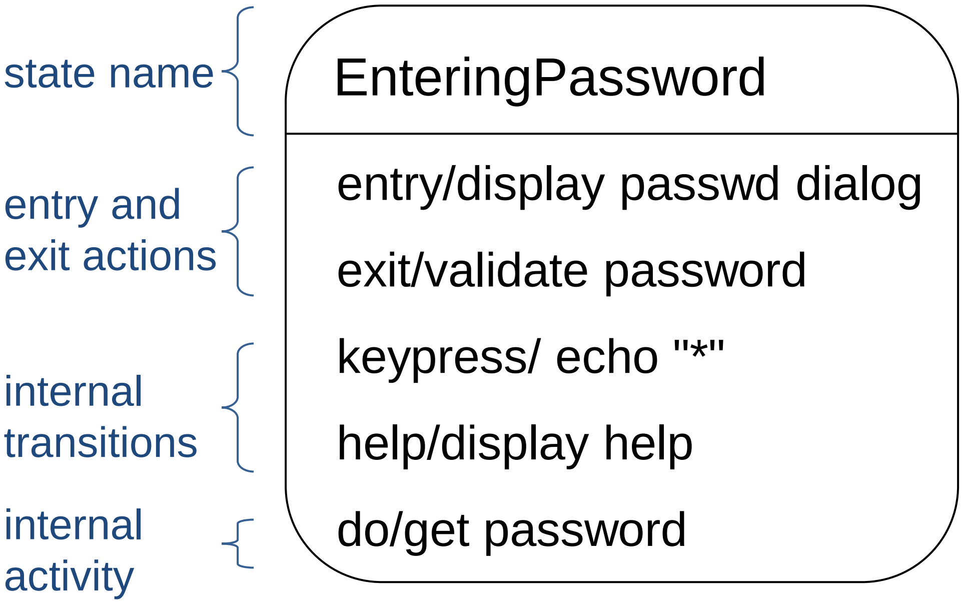
How many states?



State syntax

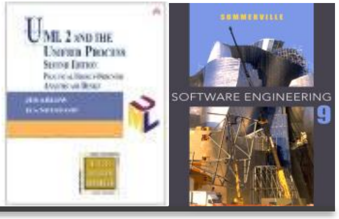


- Actions are **instantaneous** and **uninterruptible**
  - Entry actions occur immediately on state entry
  - Exit actions occur immediately on state leaving
- Internal transitions occur **within** the state. They do not fire transition to a new state
- Activities take a finite amount of time and are interruptible



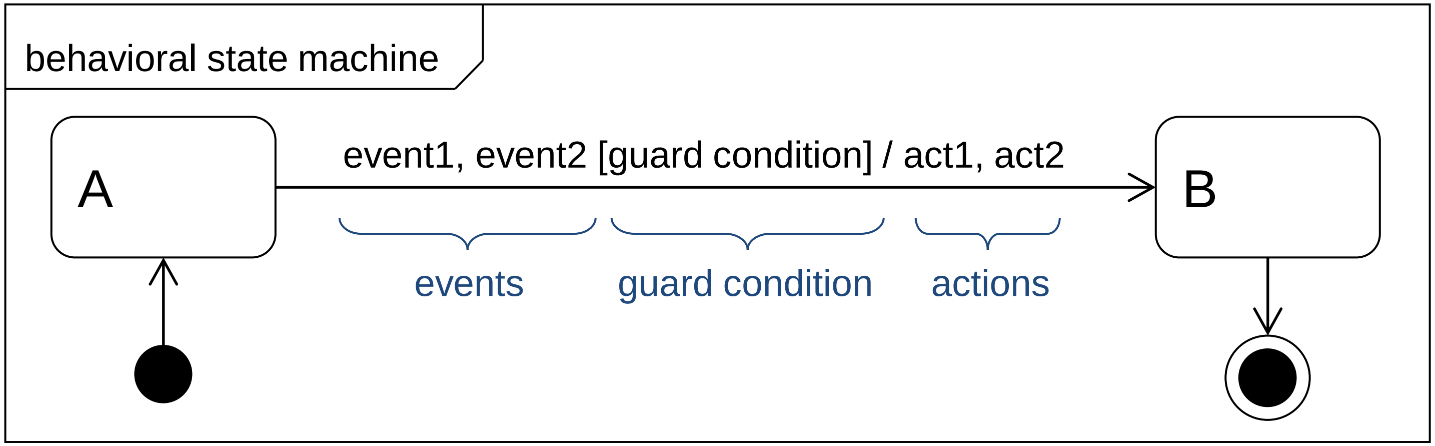
Action syntax: eventTrigger / action
Activity syntax: do / activity

Transitions



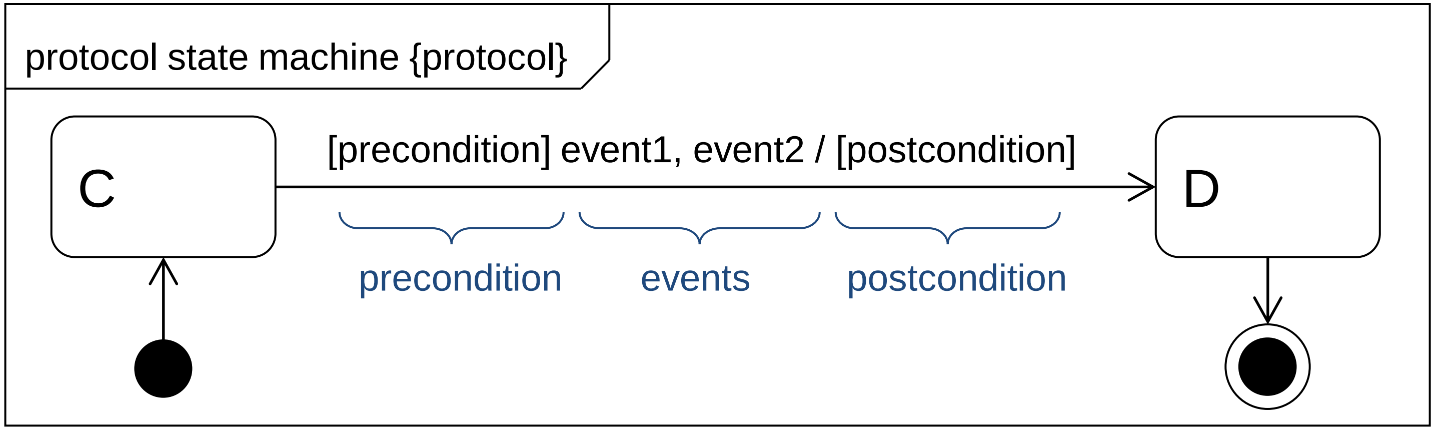
Behavioral state machine

Specifies object's reactions to events.



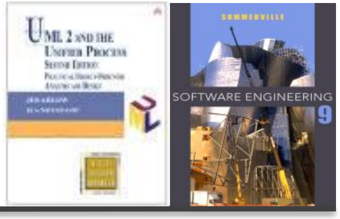
Protocol state machine

Specifies legal sequences of events.

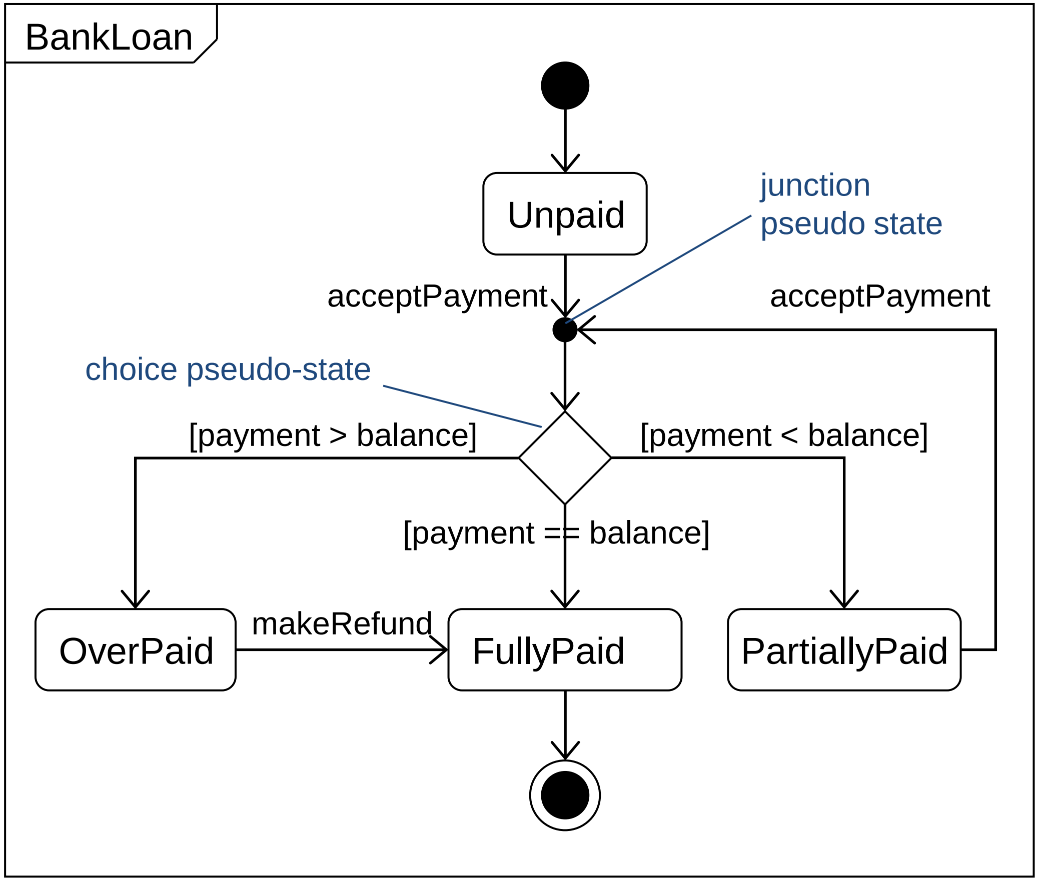


protocol state machine {protocol}
C
[precondition] event1, event2 / [postcondition]
D
precondition
events
postcondition

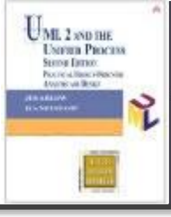
Choice and junction pseudo states



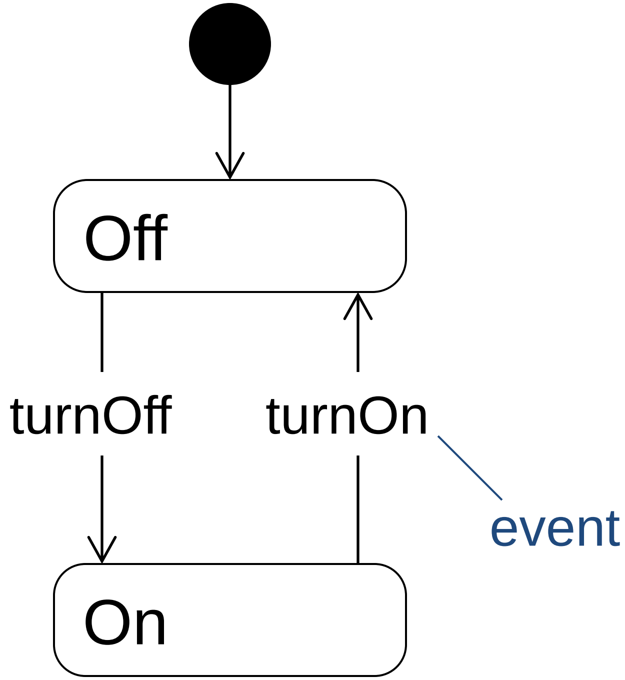
- **Choice pseudo state** directs its single incoming transition to one of its outgoing transitions
  - Each outgoing transition must have a mutually exclusive guard condition
  - Equivalent to two outgoing transitions from one state
- **Junction pseudo state** connects multiple incoming transitions into one (or more) transitions.
  - When there are more outgoing transitions, they must have guard conditions



Events

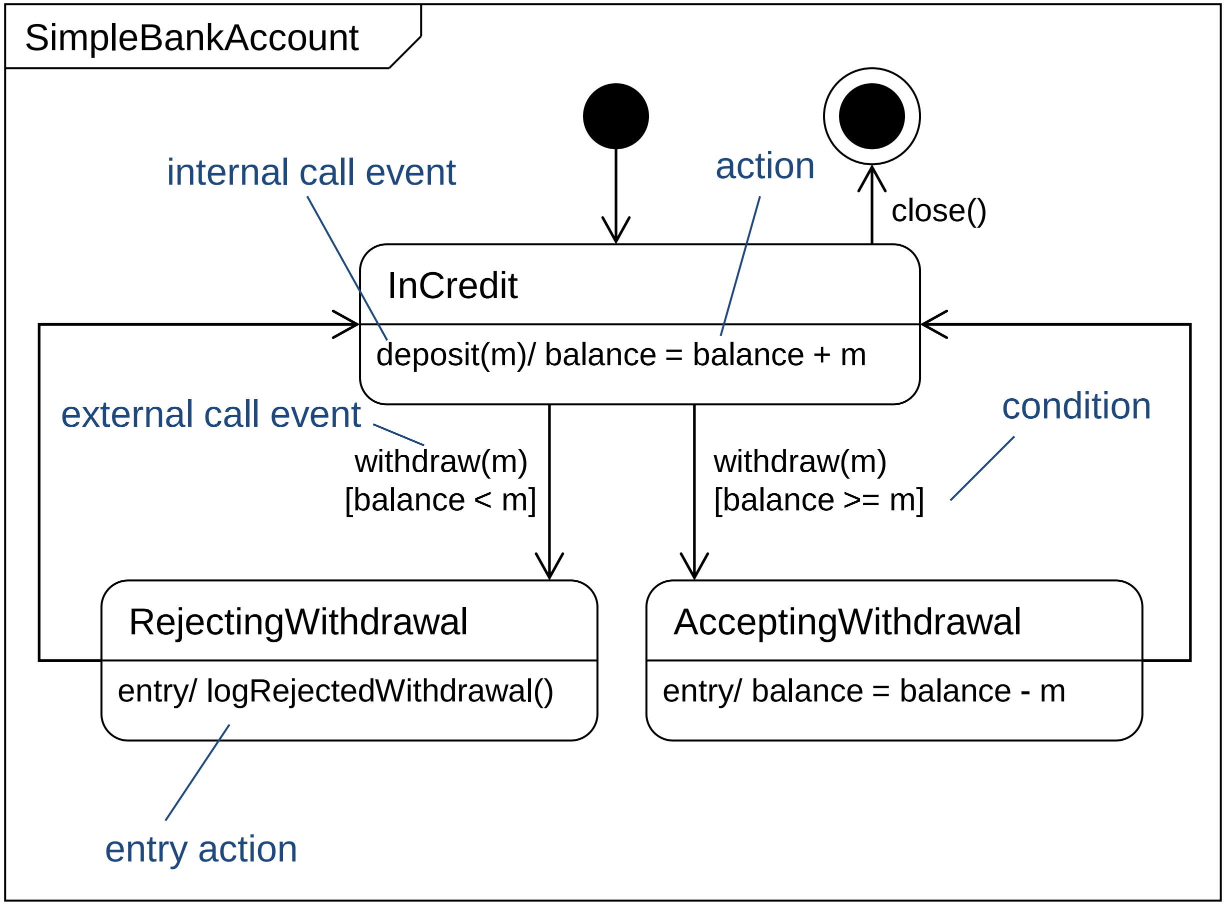


- "The specification of a noteworthy occurrence that has location in time and space"
- Events trigger transitions in state machines
- Events can be shown externally, on transitions, or internally within states (internal transitions)
- There are four types of event:
  - Call event
  - Signal event
  - Change event
  - Time event

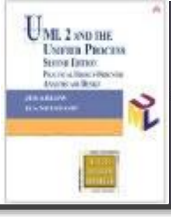


Call event

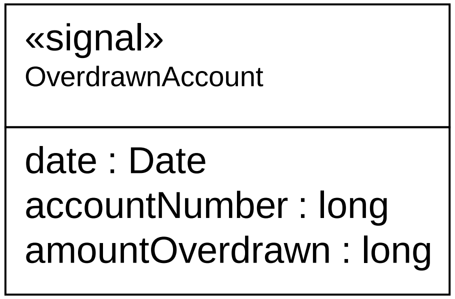
- A call for an operation execution
- The event should have the same signature as an operation of the context class
- A sequence of actions may be specified for a call event - they may use attributes and operations of the context class
- The return value must match the return type of the operation



Signal events

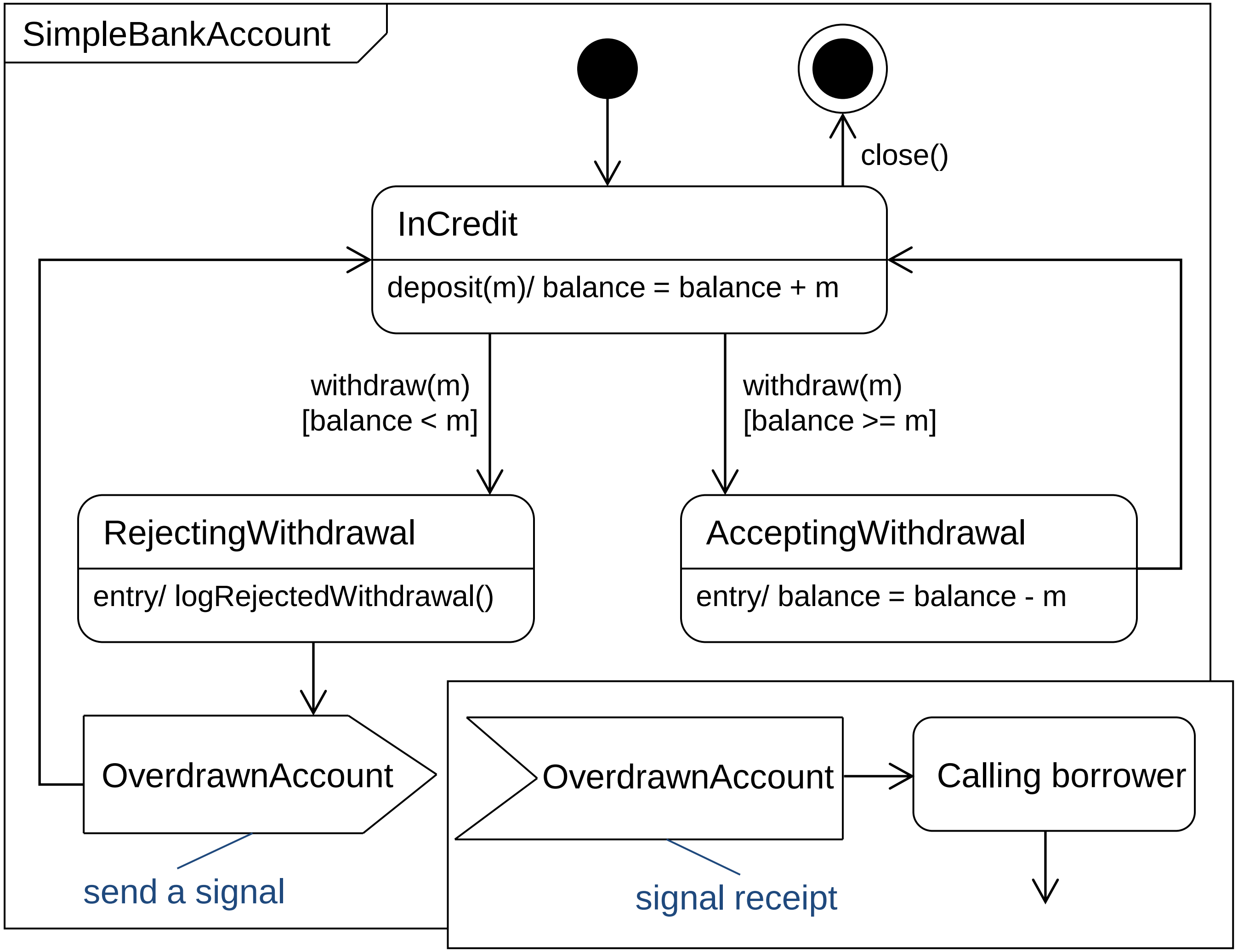


- A signal is a package of information that is sent asynchronously between objects



«signal»
OverdrawnAccount

date : Date
accountNumber : long
amountOverdrawn : long



SimpleBankAccount

close()

InCredit

deposit(m)/ balance = balance + m

withdraw(m)
[balance < m]

withdraw(m)
[balance >= m]

RejectingWithdrawal

entry/ logRejectedWithdrawal()

AcceptingWithdrawal

entry/ balance = balance - m

OverdrawnAccount

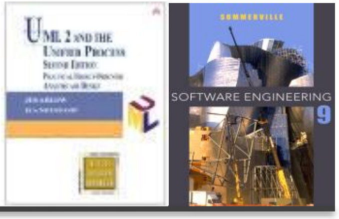
OverdrawnAccount

Calling borrower

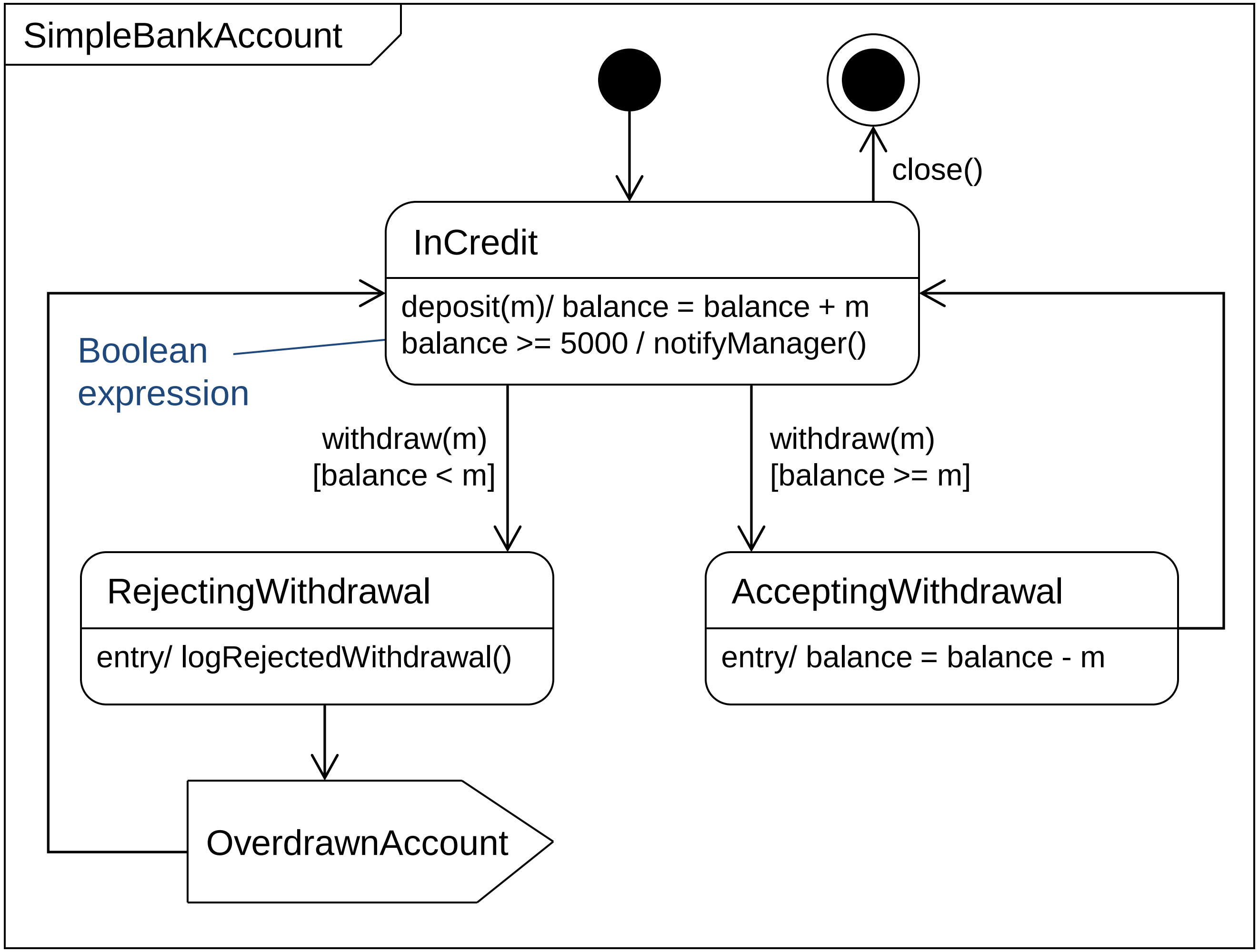
send a signal

signal receipt

Change events

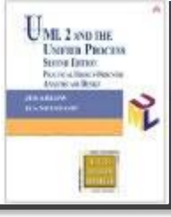


- The action is performed when the Boolean expression transitions from false to true
  - The event is **edge triggered** on a false to true transition
  - The values in the Boolean expression must be constants, globals or attributes of the context class
- A change event implies continually testing the condition whilst in the state

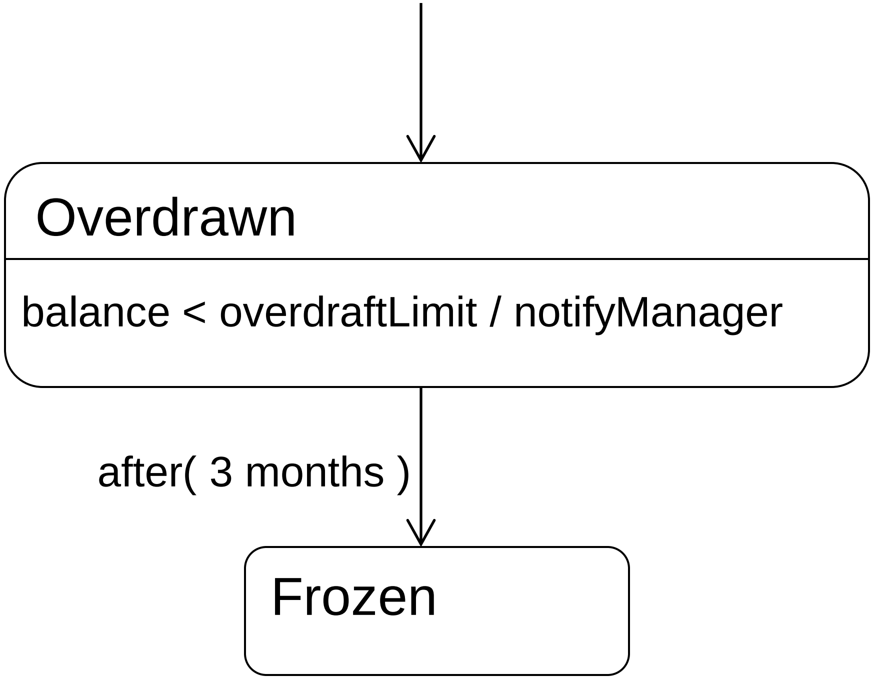


SimpleBankAccount
close()
InCredit
deposit(m)/ balance = balance + m
balance >= 5000 / notifyManager()
Boolean expression
withdraw(m) [balance < m]
withdraw(m) [balance >= m]
RejectingWithdrawal
entry/ logRejectedWithdrawal()
AcceptingWithdrawal
entry/ balance = balance - m
OverdrawnAccount

Time events

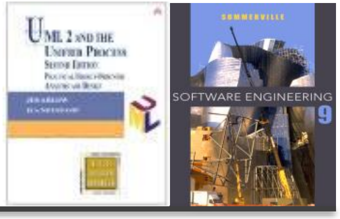


- Time events occur when a time expression becomes true
- There are two keywords, **after** and **when**
- Elapsed time:
  - after(3 months)
- Absolute time:
  - when(date =20/3/2000)

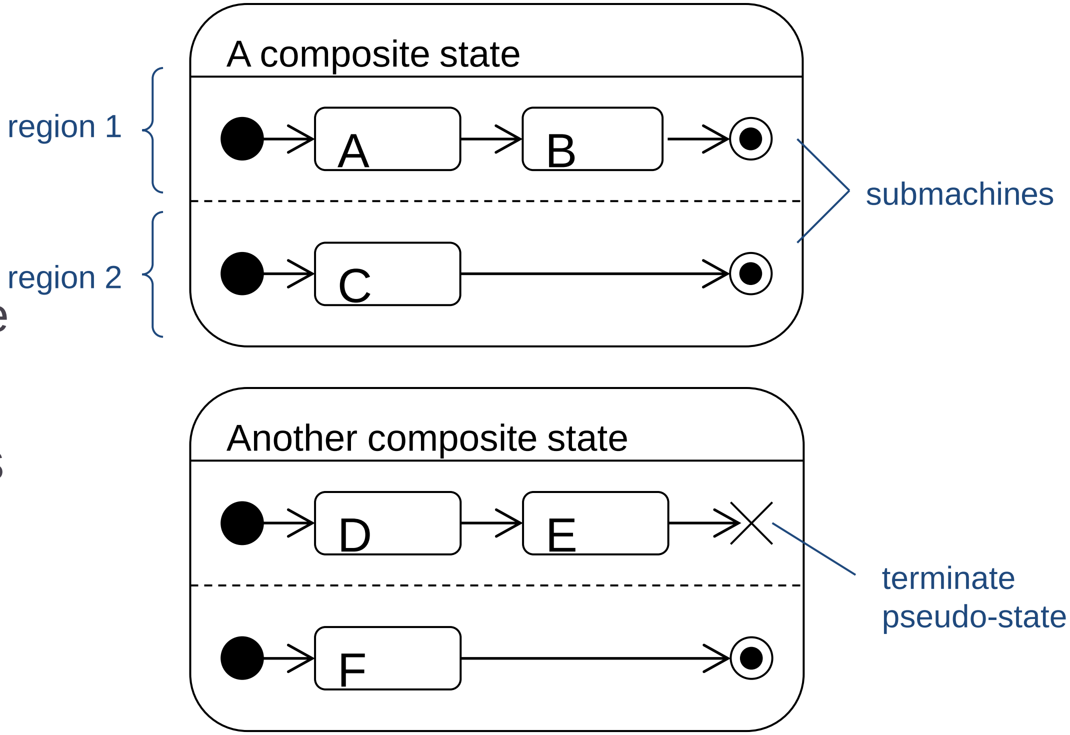


Context: CreditAccount class

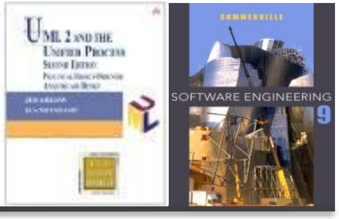
Composite states



- ✧ Have one or more regions that each contain a nested submachine
  - ▪ Simple composite state
    - exactly one region
  - ▪ Orthogonal composite state
    - two or more regions
- ✧ The final state terminates its enclosing region – all other regions continue to execute
- ✧ The terminate pseudo-state terminates the whole state machine

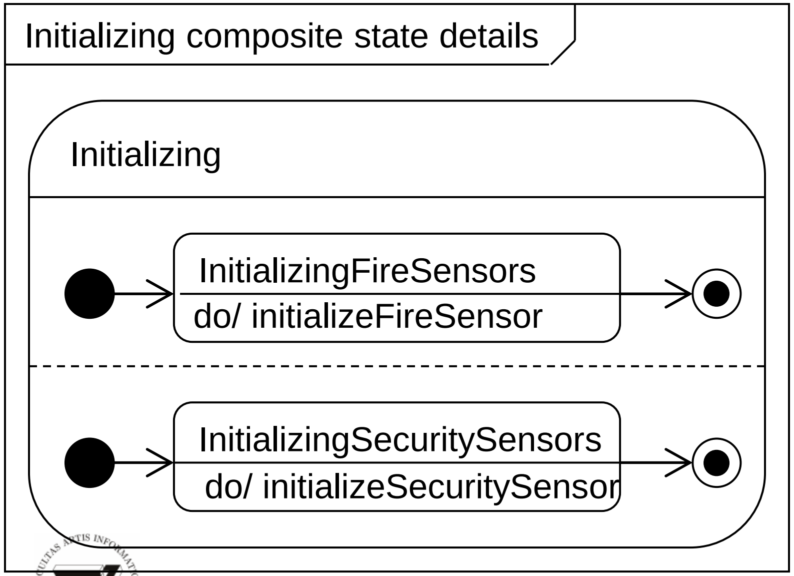


Orthogonal composite states



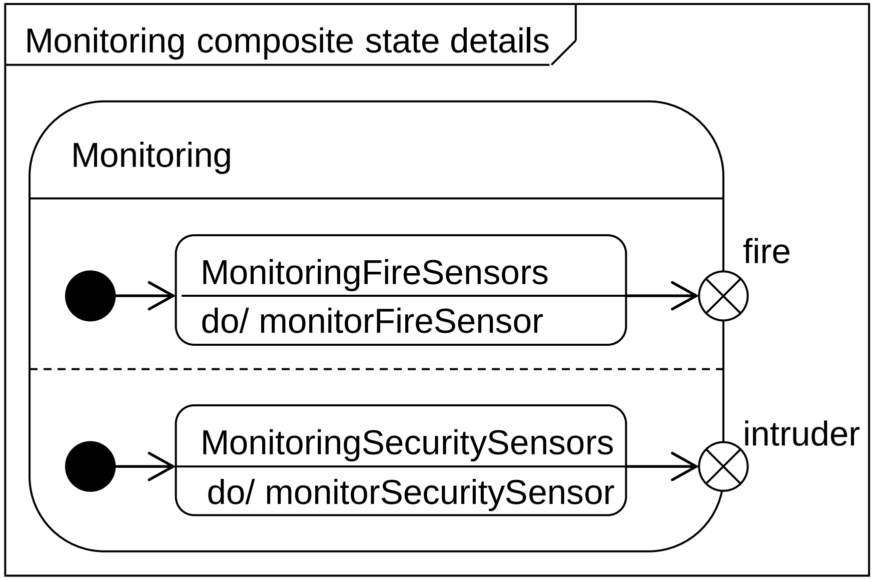
- ✧ Has two or more regions
- ✧ When we enter the superstate, both submachines start executing concurrently - this is an implicit fork

Synchronized exit - exit the superstate when *both* regions have terminated



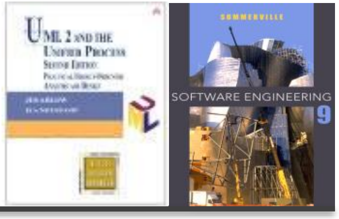
Initializing composite state details
Initializing
InitializingFireSensors
do/ initializeFireSensor
InitializingSecuritySensors
do/ initializeSecuritySensor

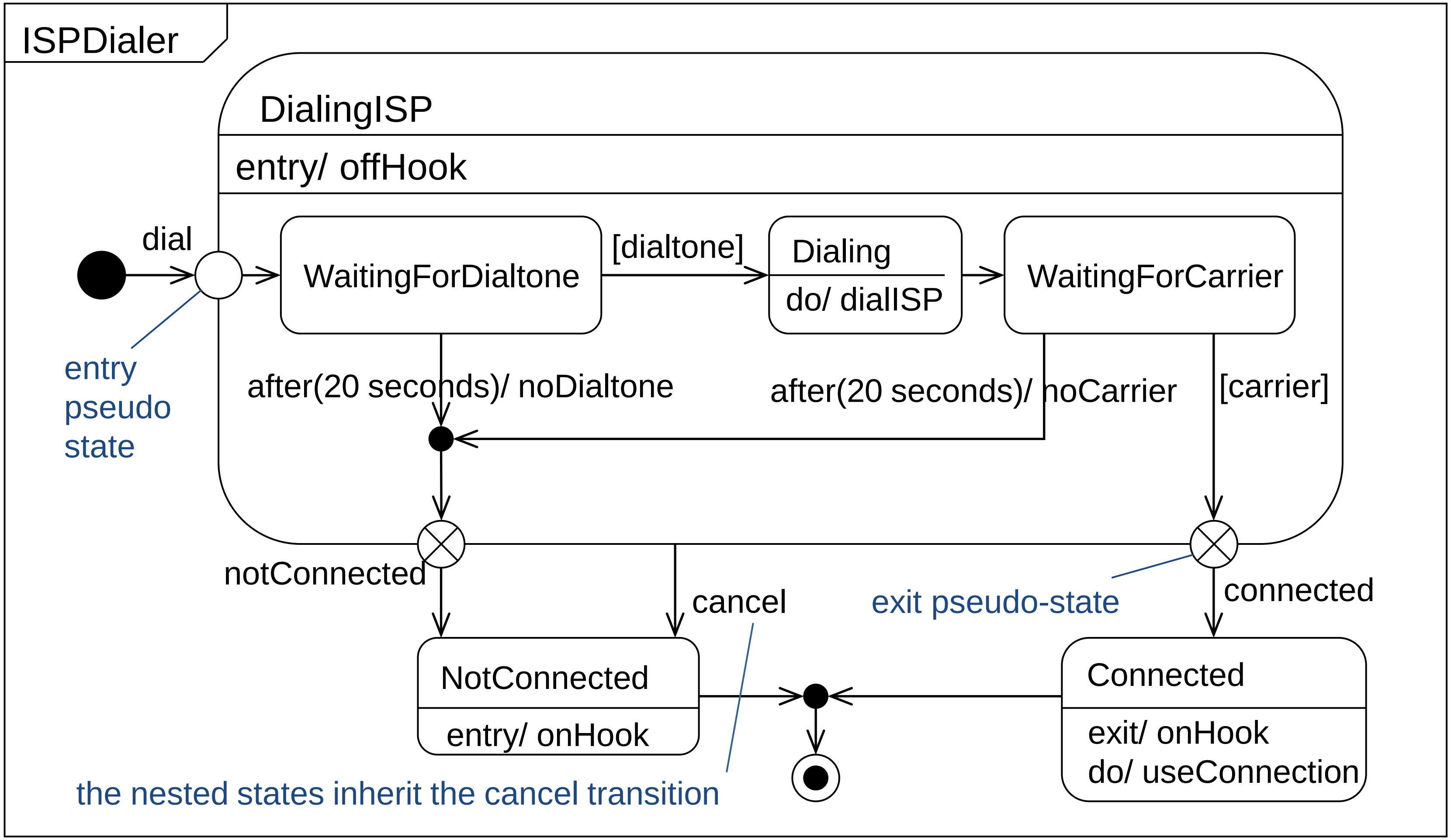
Unsynchronized exit - exit the superstate when *either* region terminates. The other region continues



Monitoring composite state details
Monitoring
MonitoringFireSensors
do/ monitorFireSensor
MonitoringSecuritySensors
do/ monitorSecuritySensor
fire
intruder

Simple composite states



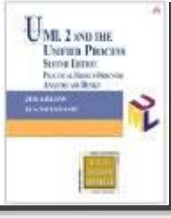


entry pseudo state

exit pseudo-state

the nested states inherit the cancel transition

Key points



- ✧ Behavioral and protocol state machines
- ✧ States
  - Initial and final
  - Exit and entry actions, activities
- ✧ Transitions
  - Guard conditions, actions
- ✧ Events
  - Call, signal, change and time
- ✧ Composite states
  - Simple and orthogonal composite states