

# DEVELOPMENT OF INFORMATION SYSTEMS

## **Lecture 8**


## **Team Software Development**

# Review of the last lecture

## Inheritance in OOP


**Inheritance** provides code **re-usability**; one of the key concepts of OOP. It represents real-world relationships between objects.

Inheritance is the procedure in which one class inherits the attributes and methods of another class.

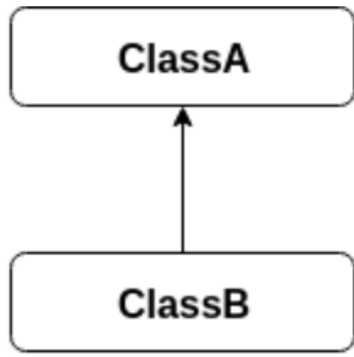
A series of three parallel white diagonal lines in the bottom right corner of the slide.

# Inheritance in OOP

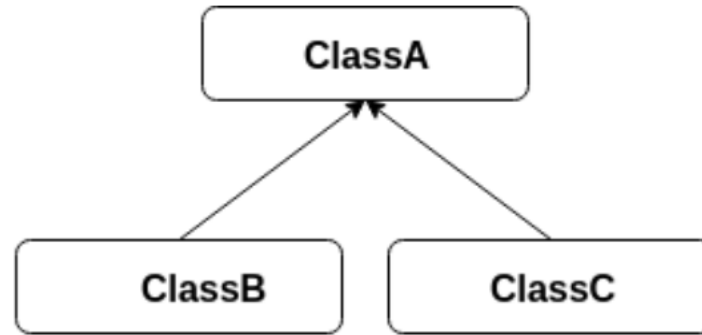
## General types of inheritance:

- Single Inheritance
  - Multiple Inheritance
  - Multi-level Inheritance
  - Hierarchical Inheritance
  - Hybrid Inheritance
- 
- A decorative graphic consisting of several parallel white lines of varying lengths, slanted diagonally upwards from left to right, located in the bottom right corner of the slide.

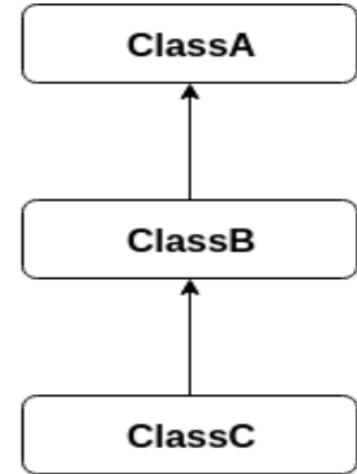
# General types of inheritance



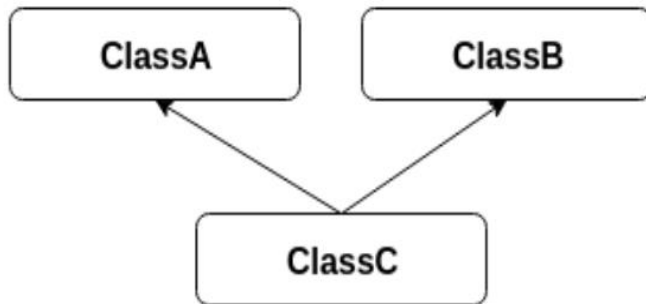
**Single Inheritance**



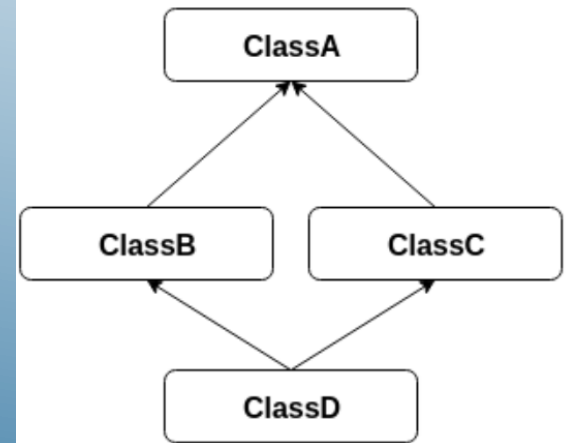
**Hierarchical inheritance**



**Multilevel Inheritance**



**Multiple Inheritance**



**Hybrid inheritance**

# Review of the last lecture

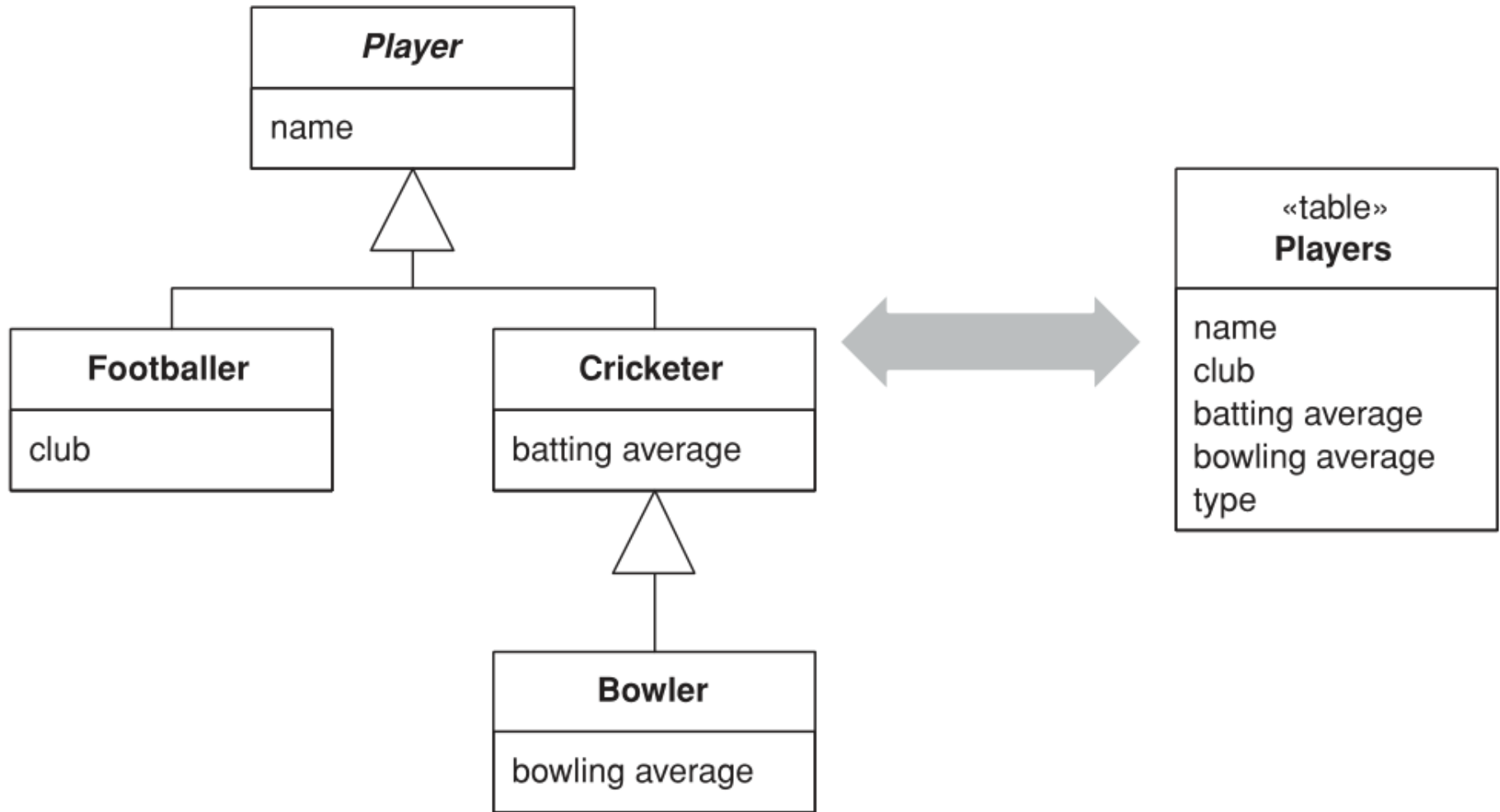
Inheritance mapping is a process to map inheritance relationships between objects from the domain model into physical storage.

Inheritance mappings patterns:

- **Single Table Inheritance**
  - **Class Table Inheritance**
  - **Concrete Table Inheritance**
  - **Inheritance Mappers**
- 

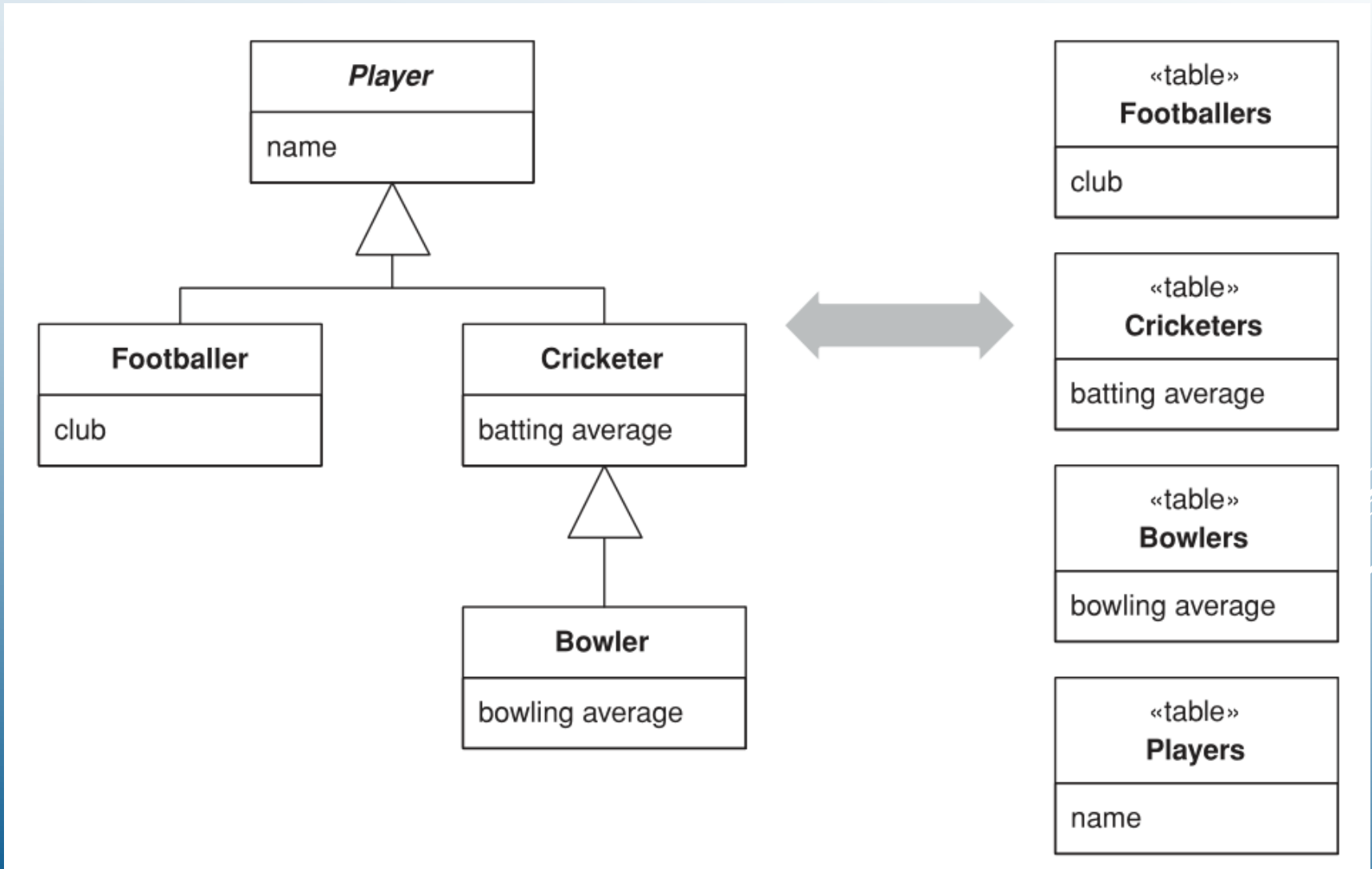
# Review - Single Table Inheritance

Single Table Inheritance maps all fields of all classes of an inheritance structure into a single table.



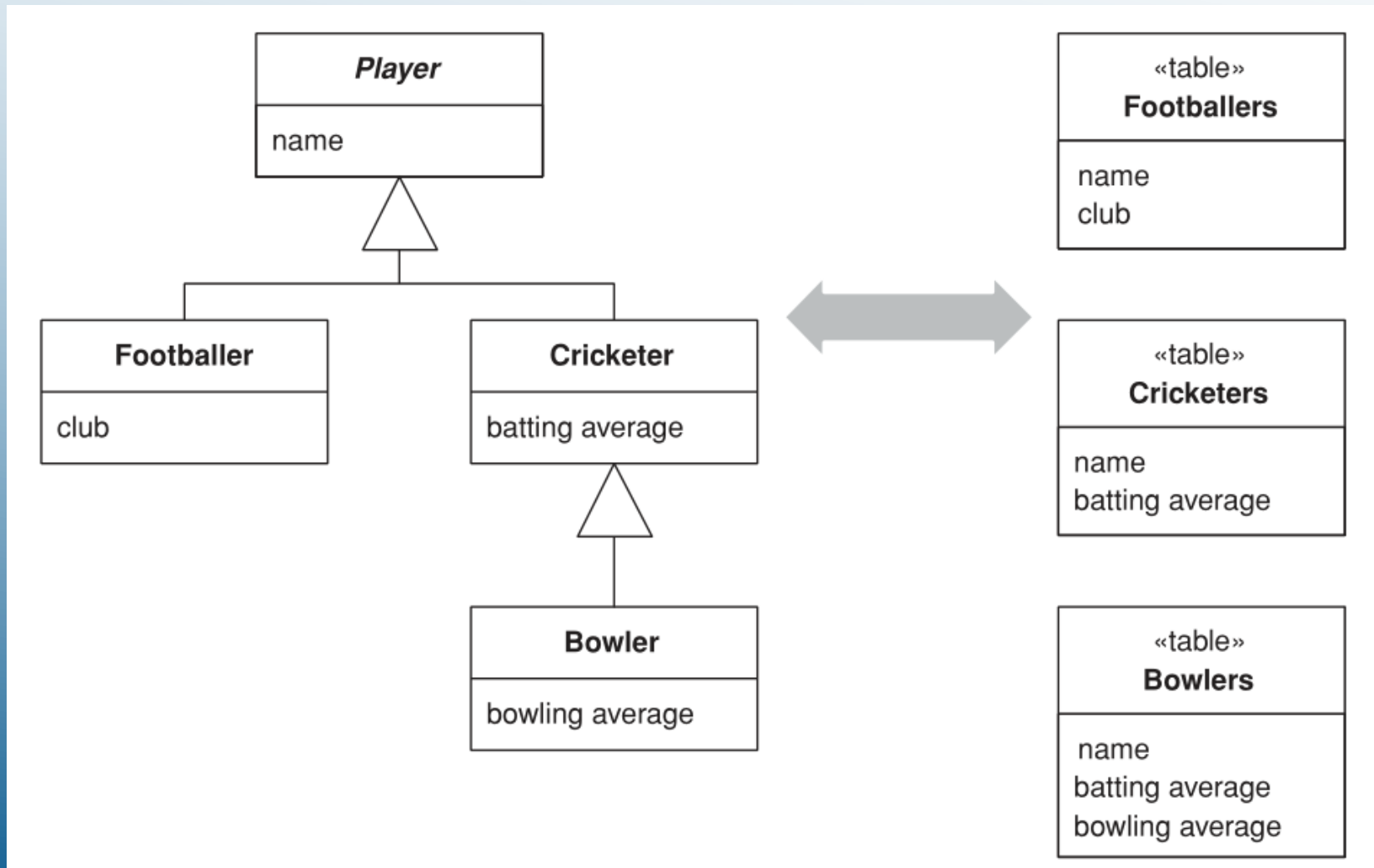
# Review - Class Table Inheritance

Represents an inheritance hierarchy of classes with one table for each class.



# Review - Concrete Table Inheritance

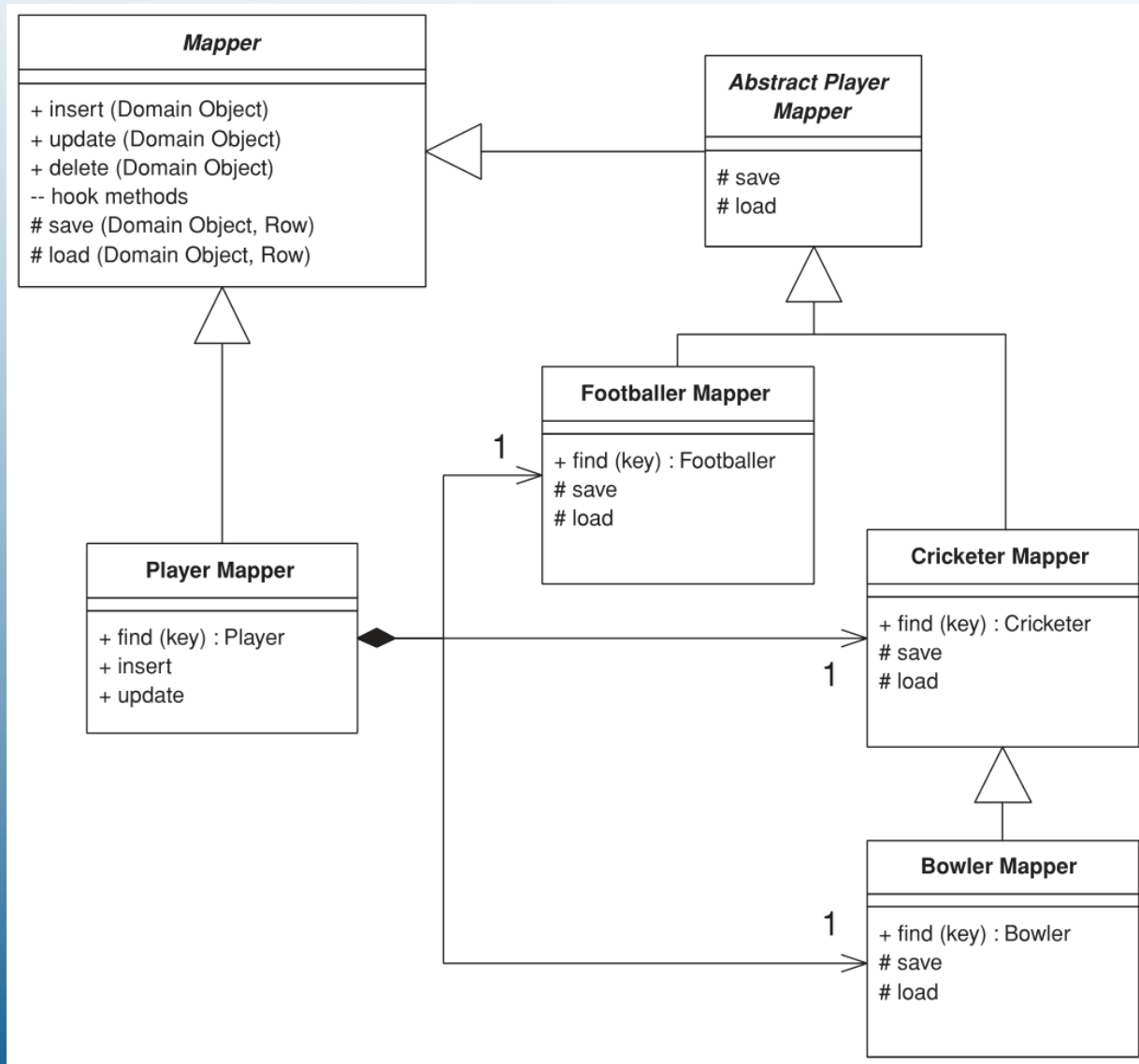
Represents an inheritance hierarchy of classes with one table per concrete class in the hierarchy.





# Review - Inheritance mappers

A structure to organize database mappers that handle inheritance hierarchies the same for all inheritance patterns.



# Software development

Software development is the process of

- conceiving
- specifying
- designing
- programming
- documenting
- testing
- bug fixing

involved in creating and maintaining applications, frameworks, or other software components

# What is needed and what is essential?

- People and their cooperation
- Plans, rules, processes, management
- Documentation
- Techniques and technologies
- Long time

**The goal is the product (software) and its quality, measured by multiple factors.**

# From assignment to product

- **People, their quality and the quality of their outputs are fundamental.**
- **It is necessary to be able to organize work and coordinate people in a team.**  
*The question is, to what extent of administrative burden.*
- **Documentation is necessary, those who do not document do not value their own work.**  
*The question is to what extent amount.*
- **A plan (time and cost) is required.**  
*It is questionable whether it is possible to estimate everything in advance.*
- **Good decisions are needed at the beginning (technology, architecture).**  
*The question is whether we seem to have always enough information.*

# Program life cycle

1. Conception
2. Requirements  
gathering/exploration/modeling
3. Design
4. Coding and debugging
5. Testing
6. Release
7. Maintenance/software evolution
8. Retirement

Your program may compress some of these steps, or combine two or more steps into a single piece of work, but all programs go through all steps of the life cycle.

# The Four Variables of Software development

1. **Cost** is probably the most constrained; you can't spend your way to quality or being on schedule, and as a developer you have very limited control over cost.

- Cost can influence the size of the team or, less often, the types of tools available to the team.
- For small companies and startups, cost also influences the environment where the developers will work.

# The Four Variables of Software development

**2. Time** is your delivery schedule and is unfortunately many times imposed on you from the outside.

- For example, most consumer products (be they hardware or software) will have a delivery date somewhere between August and October in order to hit the holiday buying season. You can't move Christmas. If you're late, the only way to fix your problem is to drop features or lessen quality, neither of which is pretty.
- Time is also where Brooks's law (Fred Brooks in 1975) gets invoked (adding programmers to a late project just makes it later).

# The Four Variables of Software development

**3. Quality** is the number and severity of defects you're willing to release with.

- You can make short-term gains in delivery schedules by sacrificing quality, but the cost is enormous: it will take more time to fix the next release, and your credibility is pretty well shot.





# The Four Variables of Software development

**4. Features** (also called scope) are what the product actually does. This is what developers should always focus on.

- It's the most important of the variables from the customer's perspective and is also the one you as a developer have the most control over.
- Controlling scope allows you to provide managers and customers control over quality, time, and cost.
- If the developers don't have control over the feature set for each release, then they are likely to blow the schedule. This is why developers should do the estimates for software work products.

# Project management triangle

The project management triangle is used by managers to analyze or understand the difficulties that may arise during the implementation and execution of a project . All projects, regardless of their size, will have many constraints.

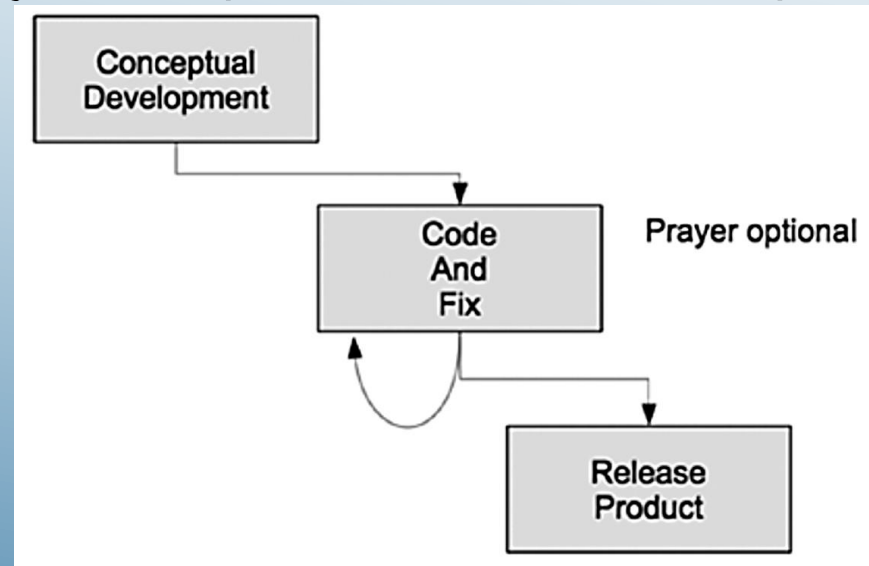


# Software development models

## Software Development Life Cycle models (SLDC)

### A Model That's not a Model At All: **Code and Fix** (Hacking)

The first model of software development we'll talk about isn't really a model at all. But it is what most of us do when we're working on small projects by ourselves, or maybe with a single partner.



It works well for proof-of-concept programs. There's no maintenance involved, and the model works well for small, single-person programs. It is, a very dangerous model for any other kind of program.

# Software development models

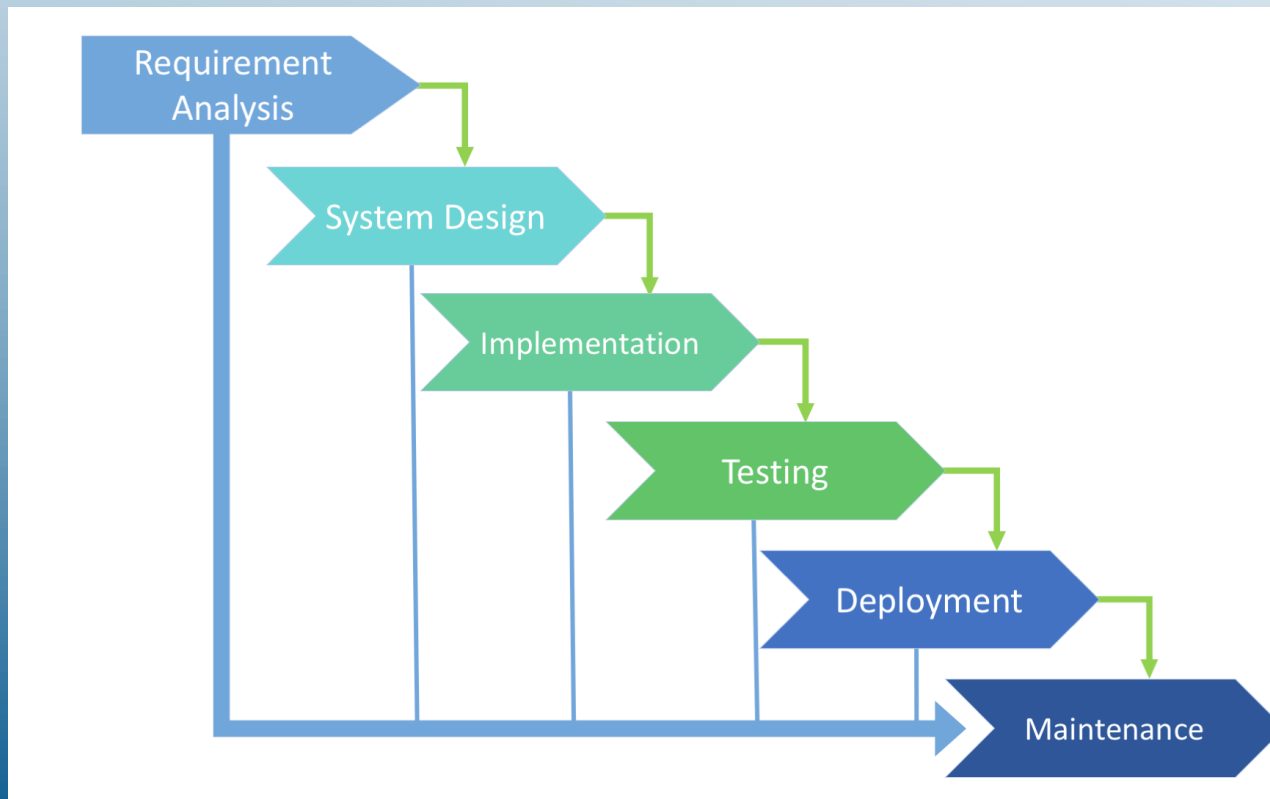
## Waterfall (1970)

The first and most traditional of the plan-driven process models and addresses all of the standard life cycle phases.

- It progresses nicely through requirements gathering and analysis, to architectural design, detailed design, coding, debugging, integration and system testing, release, and maintenance.
- It requires detailed documentation at each stage, along with reviews, archiving of the documents, sign-offs at each process phase, configuration management, and close management of the entire project.

# Waterfall

- All phases are non overlapped.
- It generally requires that you finish phase N before you continue on to phase N+1.
- You must nail down all your requirements before you start your architectural design, and finish your coding and debugging before you start anything but unit testing.



# Waterfall model properties

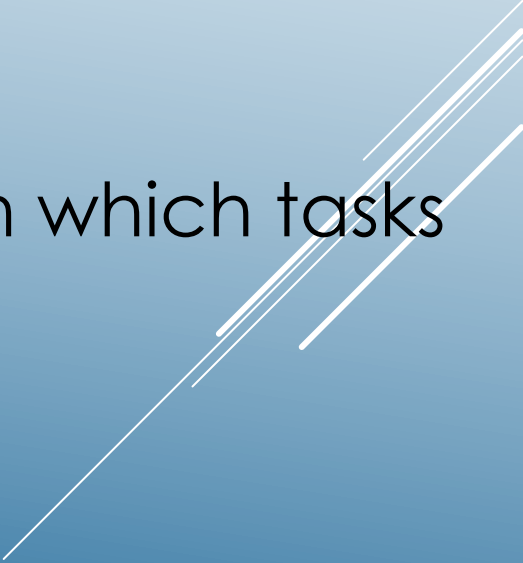
**Advantages** (plan and prepare in a careful, thorough manner before taking action)

- Early detection of errors (leads to savings).
- High emphasis on documentation (seamless replacement of people).
- Simplicity for management (project stability).

## Disadvantages

- It is impossible to perfect one phase of the software product life cycle before, before moving on to the next phase.
- It is insisting on decisions that may later turn out to be wrong.

# Aspects of teamwork

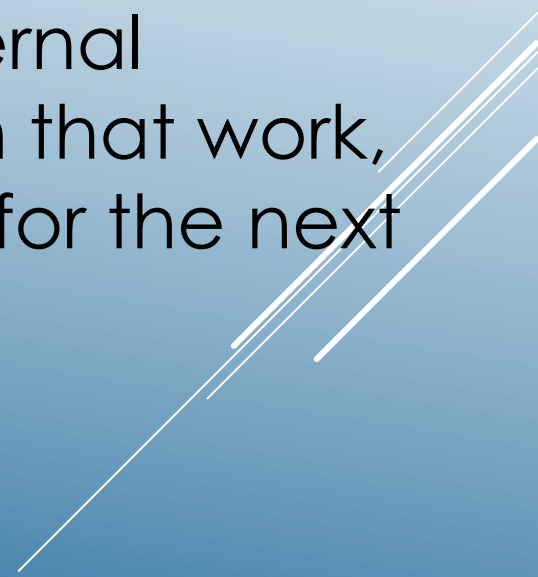
- It only takes two people to agree on something.
  - There is a difference between working in a big.  
• and a small team.
  - Robust versus Agile (nimble, active, eager) approach to a project.
  - We always have to define roles and responsibilities from which the way in which tasks are handled.
- 



# Iterative Models

The best practice is to iterate and deliver incrementally, treating each iteration as a closedend “mini-project,” including complete requirements, design, coding, integration, testing, and internal delivery.

On the iteration deadline, deliver the (fully-tested, fully-integrated) system thus far to internal stakeholders. Solicit their feedback on that work, and fold that feedback into the plan for the next iteration.

Several thin, white, parallel diagonal lines are positioned in the bottom right corner of the slide, extending from the right edge towards the bottom left.

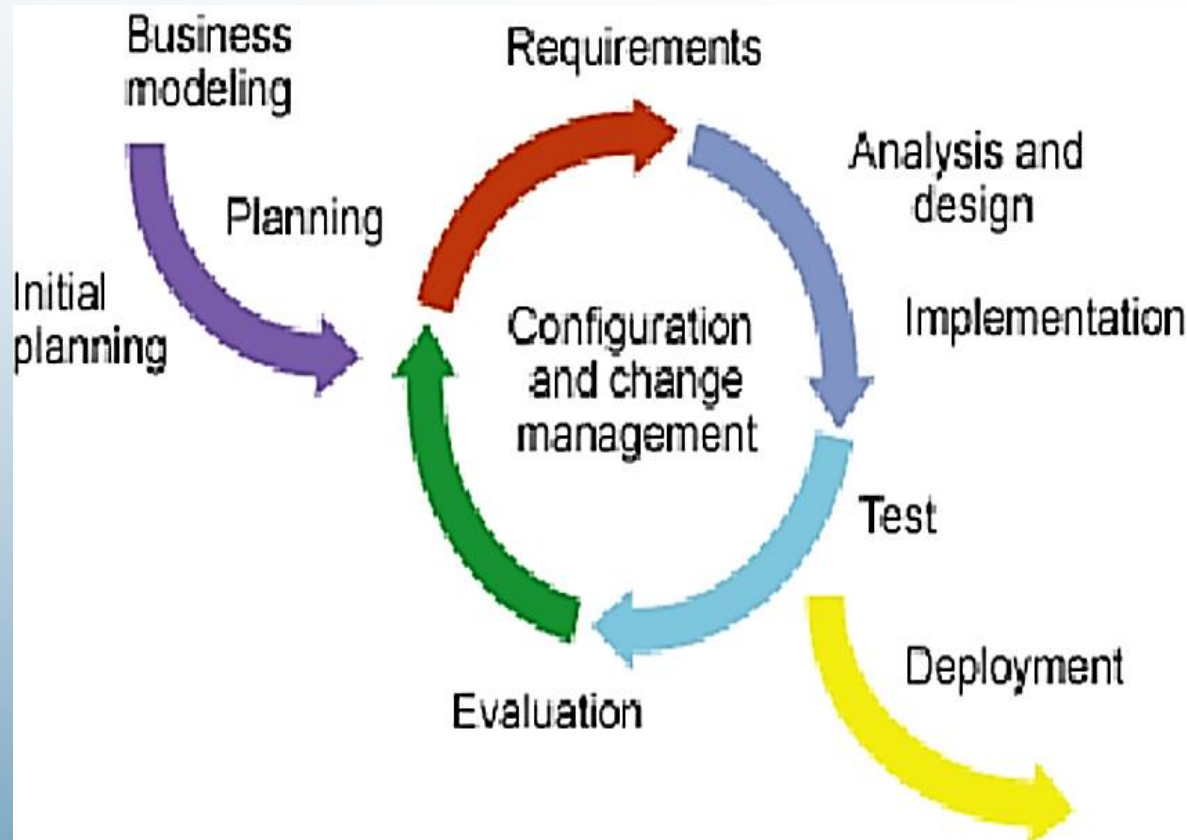


# Iterative and incremental development

- The basic principle distinguishing current approaches from the waterfall model.
- Iterative design is a methodology based on an iterative process of analysis, design, implementation (prototyping), testing and redefinition of the product.
- The incremental model is based on the principle of incrementally building a production increments based on iterative design.
- The development combines the features of the waterfall model with those of the iterative prototyping.

# Iterative and incremental development

In iterative development, each cycle of the iteration subsumes the software of the previous iteration and adds new capabilities to the evolving product to produce a next, expanded version of the software.



# Iterative and incremental development

## Advantages

- continuous integration, verification and validation of the evolving product;
- frequent demonstrations of progress;
- early detection of defects;
- early warning of process problems;
- systematic incorporation of the inevitable rework that occurs in software development;
- early delivery of subset capabilities (if desired).

## Disadvantages

- Iterative model requires more resources than the waterfall model.
- Constant management is required
- Issues with architecture or design may occur because not all the requirements are foreseen during the short planning stage
- Bad choice for the small projects
- The process is difficult to manage

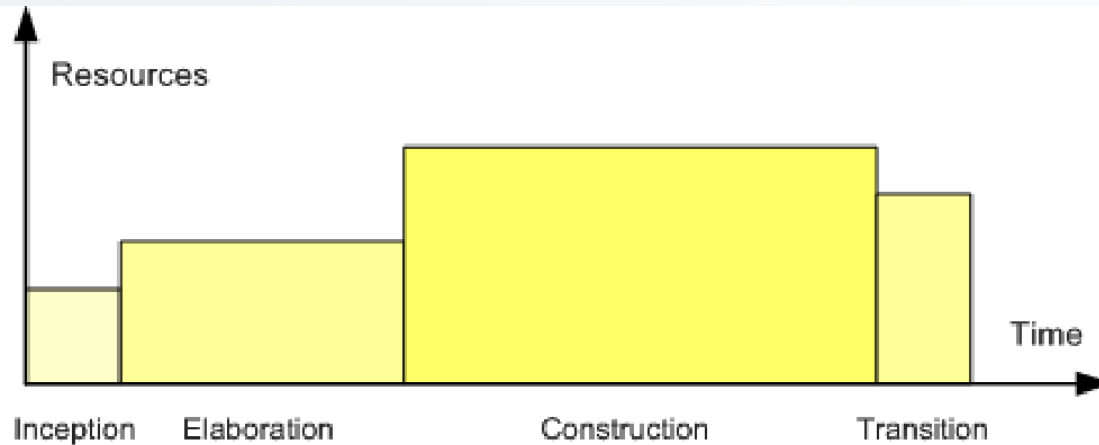
# UP: Unified Process

Ivar Jacobson (1999, Unified Software Development Process).



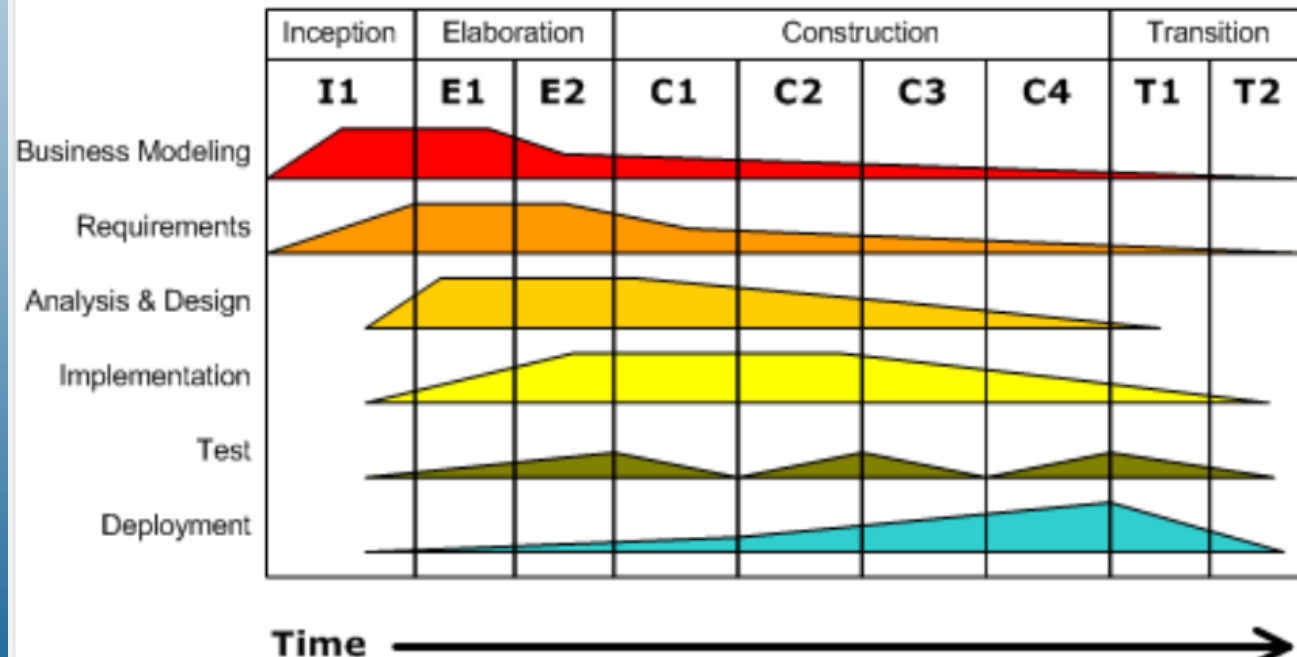
- Two key principles: iterative and incremental development.
- Three key characteristics: Use-case-driven, Architecture-centric, Risk-focused.
- Four phases: Inception, Elaboration, Construction, Transition.
- UML, documents.

# UP: Unified Process



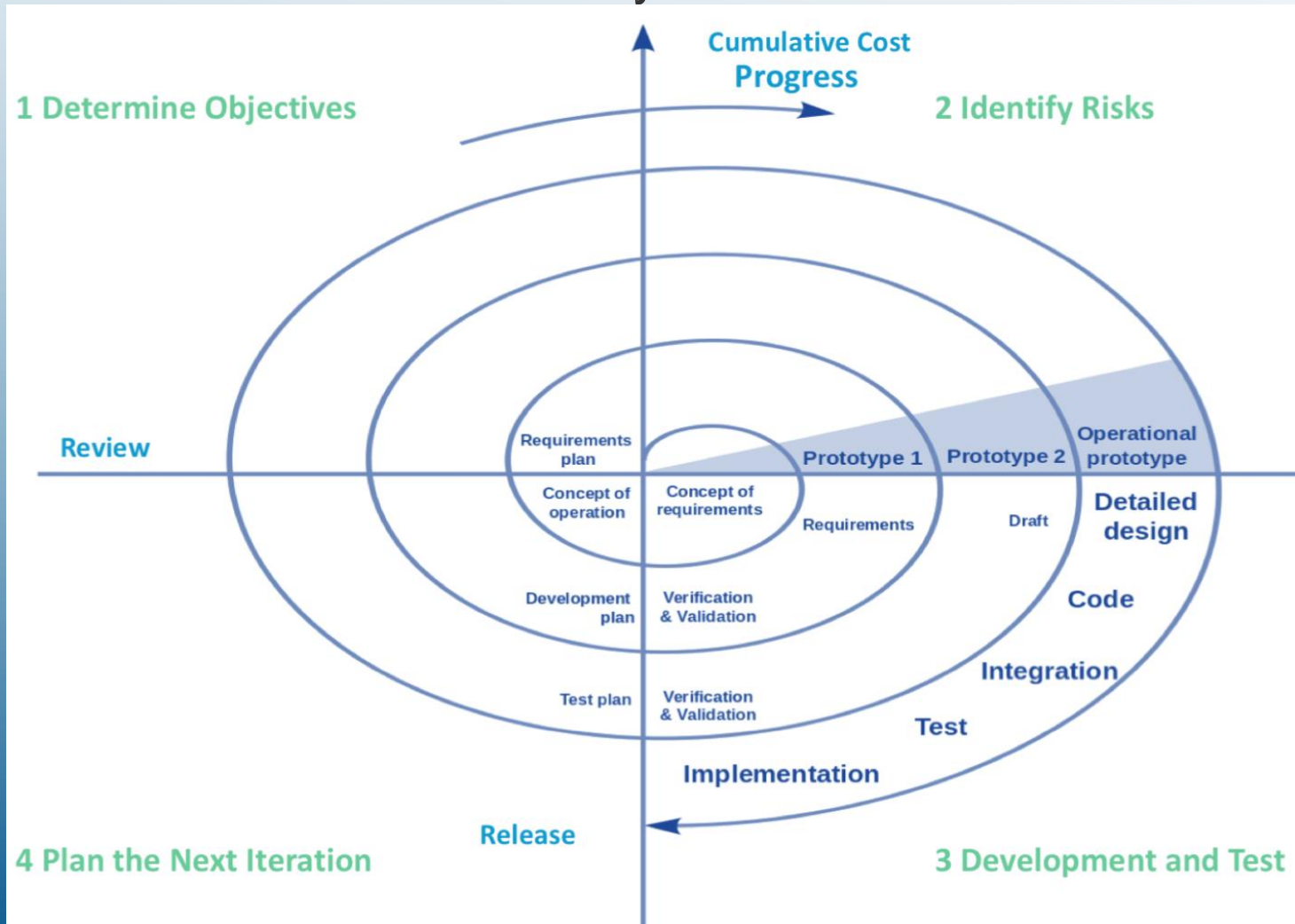
## Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.



# Spiral Model (risk-focused)

Model, which combines architecture and prototyping by stages. It is a combination of the Iterative and Waterfall models with the significant accent on the risk analysis.





# Spiral Model


- The main issue of the spiral model – is defining the right moment to make a step into the next stage.
- The preliminary set time frames are recommended as the solution to this issue.
- The shift to the next stage is done according to the plan, even if the work on the previous stage isn't done yet.
- The plan is introduced basing on the statistic data, received during the previous projects even from the personal developer's experience.

# Spiral Model

## Advantages


- Lifecycle is divided into small parts, and if the risk concentration is higher, the phase can be finished earlier to address the treats
- The development process is precisely documented yet scalable to the changes
- The scalability allows to make changes and add new functionality even at the relatively late stages
- The earlier working prototype is done - sooner users can point out the flaws

## Disadvantages

- Can be quite expensive
  - The risk control demands involvement of the highly-skilled professionals
  - Can be ineffective for the small projects
  - Big number of the intermediate stages requires excessive documentation
- 

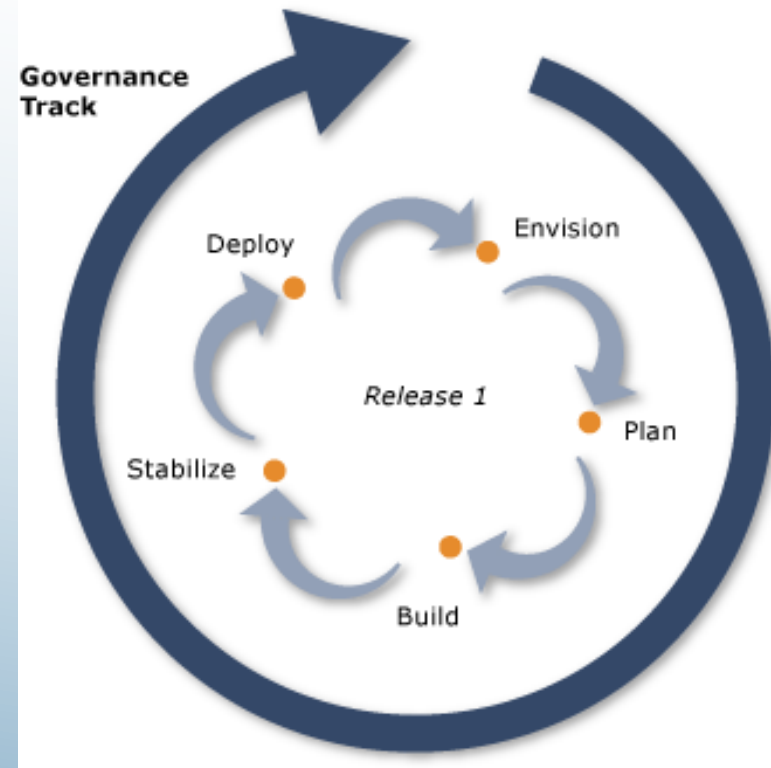


# Examples of incremental methods

- Rational Unified Process (RUP)
  - Microsoft Solutions Framework (MSF)
  - Oracle Unified Method (OUM)
  - Open Unified Process (OpenUP)
- 
- A decorative graphic consisting of several parallel white lines of varying lengths, slanted diagonally upwards from left to right, located in the bottom right corner of the slide.

# Microsoft Solution Framework

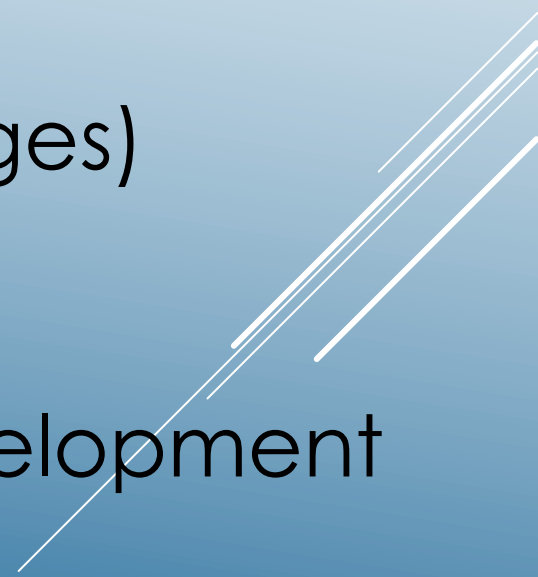
Microsoft Solutions Framework (MSF) is a set of principles, models, disciplines, concepts, and guidelines for delivering information technology services from Microsoft. MSF is not limited to developing applications only.



- Model consists of series of short development cycles and iterations. Identifying requirements, product development, and testing occur in overlapping iterations resulting in incremental completion to ensure a flow of value of the project.
- Each iteration has a different focus and result in a stable portion of the overall system.


# Microsoft Solution framework

## Key principles

- Open communication
  - Shared vision
  - Competences in the team
  - Responsibilities and their sharing
  - Incremental value
  - Be agile (adaptation to changes)
  - Quality
  - Experience
  - Customer included in the development phase.
- 
- A series of three parallel white diagonal lines are positioned in the bottom right corner of the slide, extending from the bottom edge towards the right edge.

# Agile development

## Manifesto for Agile Software Development (2001)

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



# Manifesto for Agile Software Development

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

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

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

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

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

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

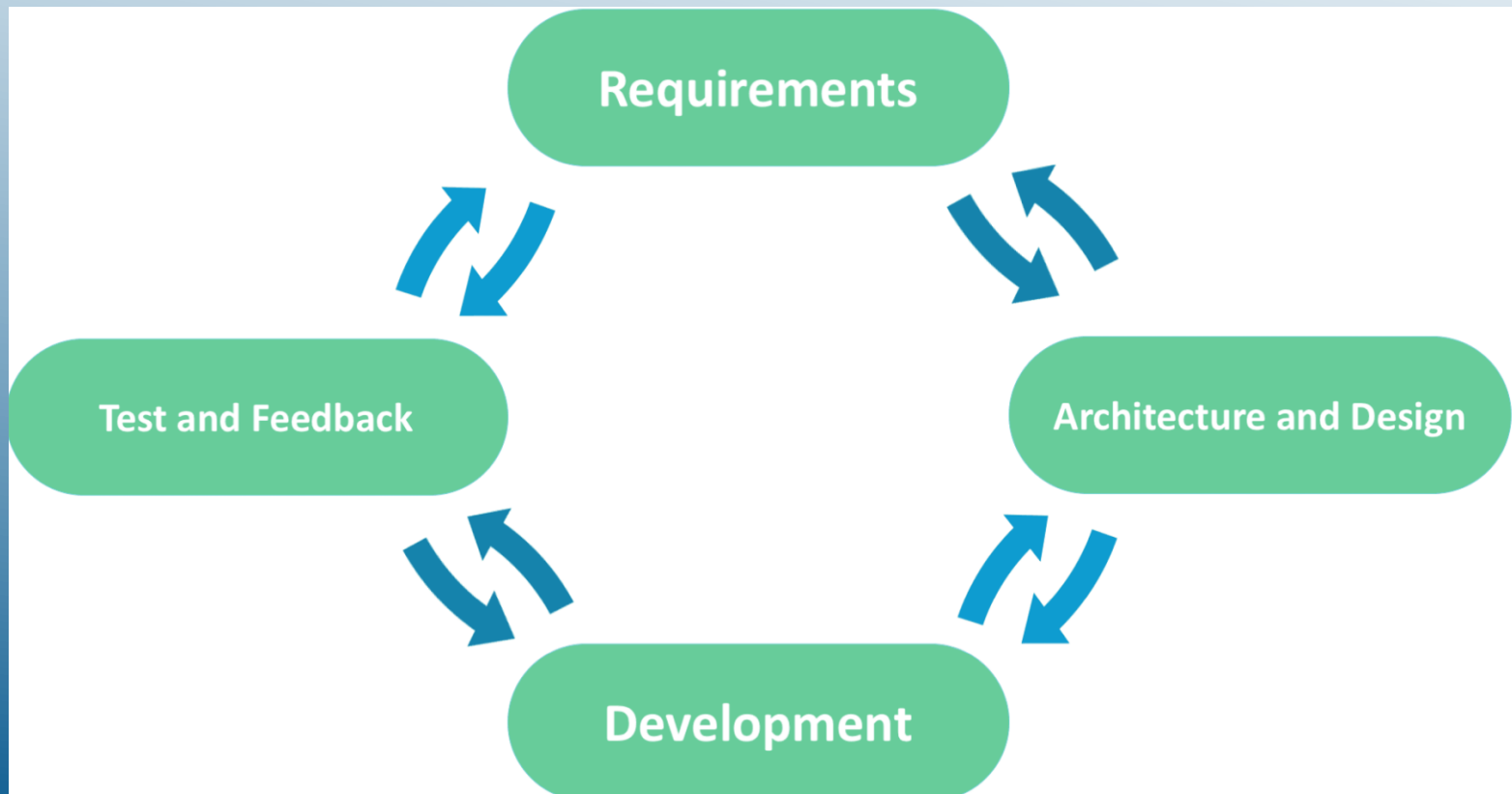
# 12 principles

1. Customer satisfaction by rapid delivery of useful software.
2. Welcome changing requirements, even late in development.
3. Working software is delivered frequently (weeks rather than months).
4. Working software is the principal measure of progress.
5. Sustainable development, able to maintain a constant pace.
6. Close, daily co-operation between business-people and developers.
7. Face-to-face conversation is the best form of communication (co-location).
8. Projects are built around motivated individuals, who should be trusted.
9. Continuous attention to technical excellence and good design.
10. Simplicity
11. Self-organizing teams
12. Regular adaptation to changing circumstances

# Agile model of development

In the agile methodology after every development iteration, the customer is able to see the result and understand if he is satisfied with it or he is not.

This is one of the advantages of the agile software development life cycle model.





# Agile model development

## Advantages

- Corrections of functional requirements are implemented into the development process to provide the competitiveness
- Project is divided by short and transparent iterations
- Risks are minimized thanks to the flexible change process
- Fast release of the first product version

## Disadvantages

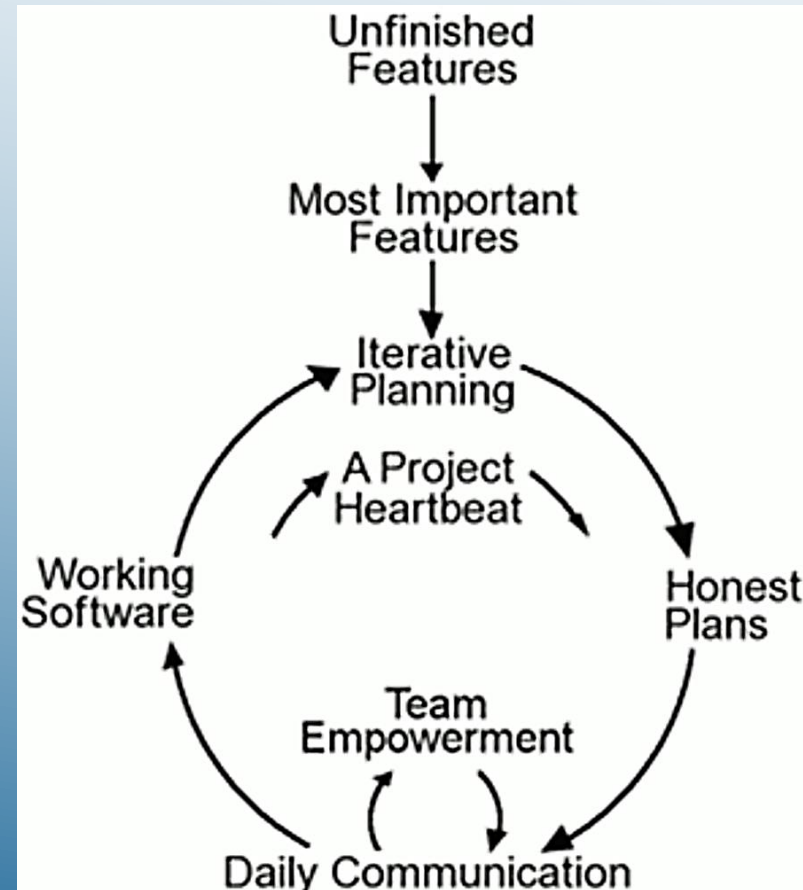
- Difficulties with measuring the final cost because of permanent changes
- The team should be highly professional and client-oriented
- New requirements may conflict with the existing architecture
- With all the corrections and changes there is possibility that the project will exceed expected time



# Extreme Programming (XP)

Extreme Programming Explained (1999) Kent Beck  
XP is one of several popular Agile Processes.

XP is successful because it stresses customer satisfaction. Instead of delivering everything you could possibly want on some date far in the future this process delivers the software you need as you need it. Extreme Programming empowers your developers to confidently respond to changing customer requirements, even late in the life cycle.



# Five values of XP

**Communication** - Transfer knowledge from one team member to everyone else on the team. Face to face discussion with the aid of a white board or other drawing mechanism.

**Simplicity** - The purpose of this is to avoid waste and do only absolutely necessary things such as keep the design of the system as simple as possible so that it is easier to maintain, support, and revise. “what is the simplest thing that will work?”

**Feedback** - Through constant feedback about their previous efforts, teams can identify areas for improvement and revise their practices. Feedback also supports simple design.

**Courage** - “effective action in the face of fear”. This definition shows a preference for action based on other principles so that the results aren’t harmful to the team. You need courage to raise organizational issues that reduce your team’s effectiveness.

**Respect** - The members of your team need to respect each other in order to communicate with each other, provide and accept feedback that honors your relationship, and to work together to identify simple designs and solutions.

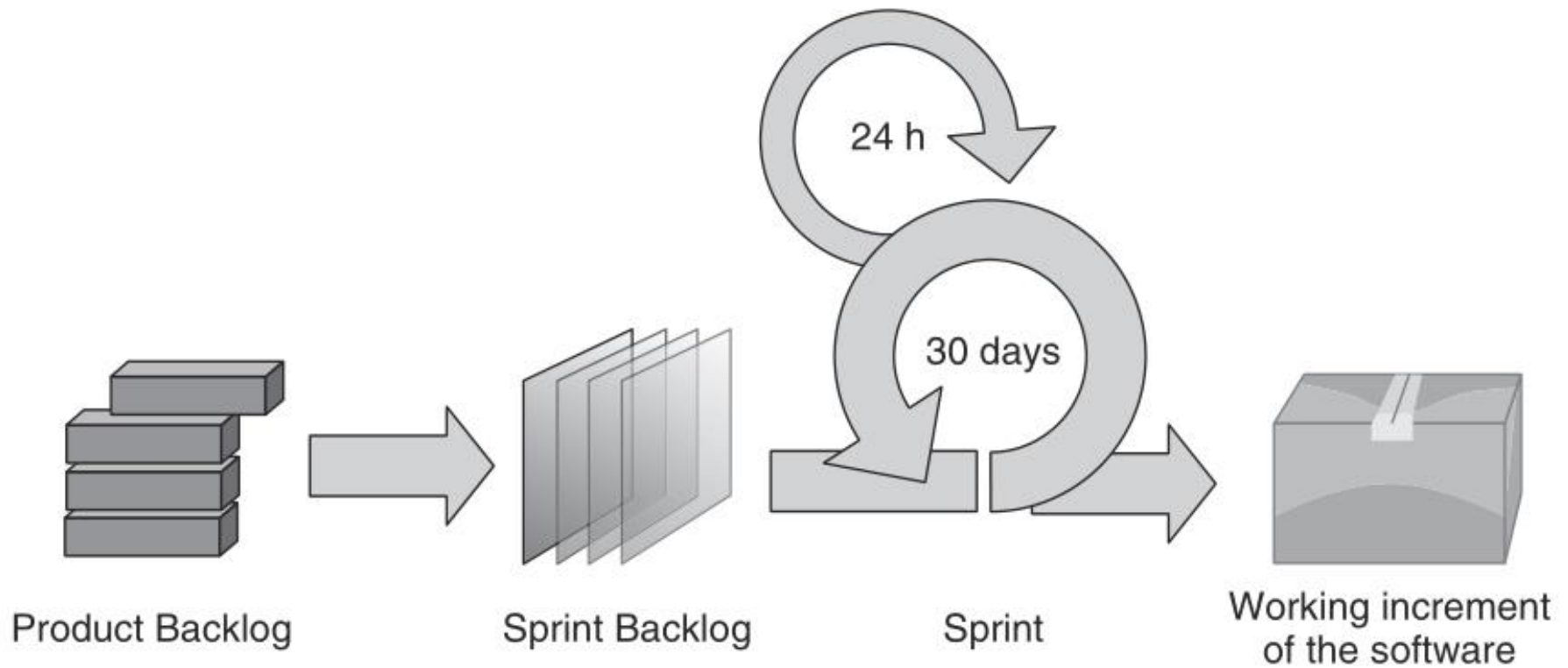
# XP practices

- **The Planning Game** (release, iteration)
- **Small Releases** (every iteration deliver to user)
- **Metaphor** (common vision of how the program works,)
- **Simple Design** (simple but always adequate design)
- **Testing**
- **Refactoring**
- **Pair Programming** (two programmers side by side)
- **Collective Ownership** (programmers can improve any code at any time)
- **Continuous Integration**
- **Sustainable pace** (work hard, and at a pace that can be sustained indefinitely)
- **On-site Customer**
- **Coding Standard**

# SCRUM model

1995, Sutherland and Schwaber

The Scrum model is a framework for planning and conducting software projects based on the principles of Agile development



# SCRUM model

**ScrumMaster** - project manager/leader. He writes **User Stories**, prioritizes them, and places them in the “ ProductBacklog. ”

**Product Owner** – the customer

**Team** – software developers. Teams include up to 10 software developers

**Sprints** – development iterations typically 30 days duration, it can be delivered to the users, if desired.

- The features to be implemented in a sprint are determined during a sprint planning meeting.
- The features to be included are derived from the Product Backlog and placed in a “ Sprint Backlog. ”
- Brief (15 minute) stand - up meetings are held **each day** during a sprint to review work accomplished the previous day and to plan the work.
- Each sprint is followed by a meeting (a “ sprint retrospective ” ) during which the Team reviews the sprint and determines how they can improve their work processes in future sprints.

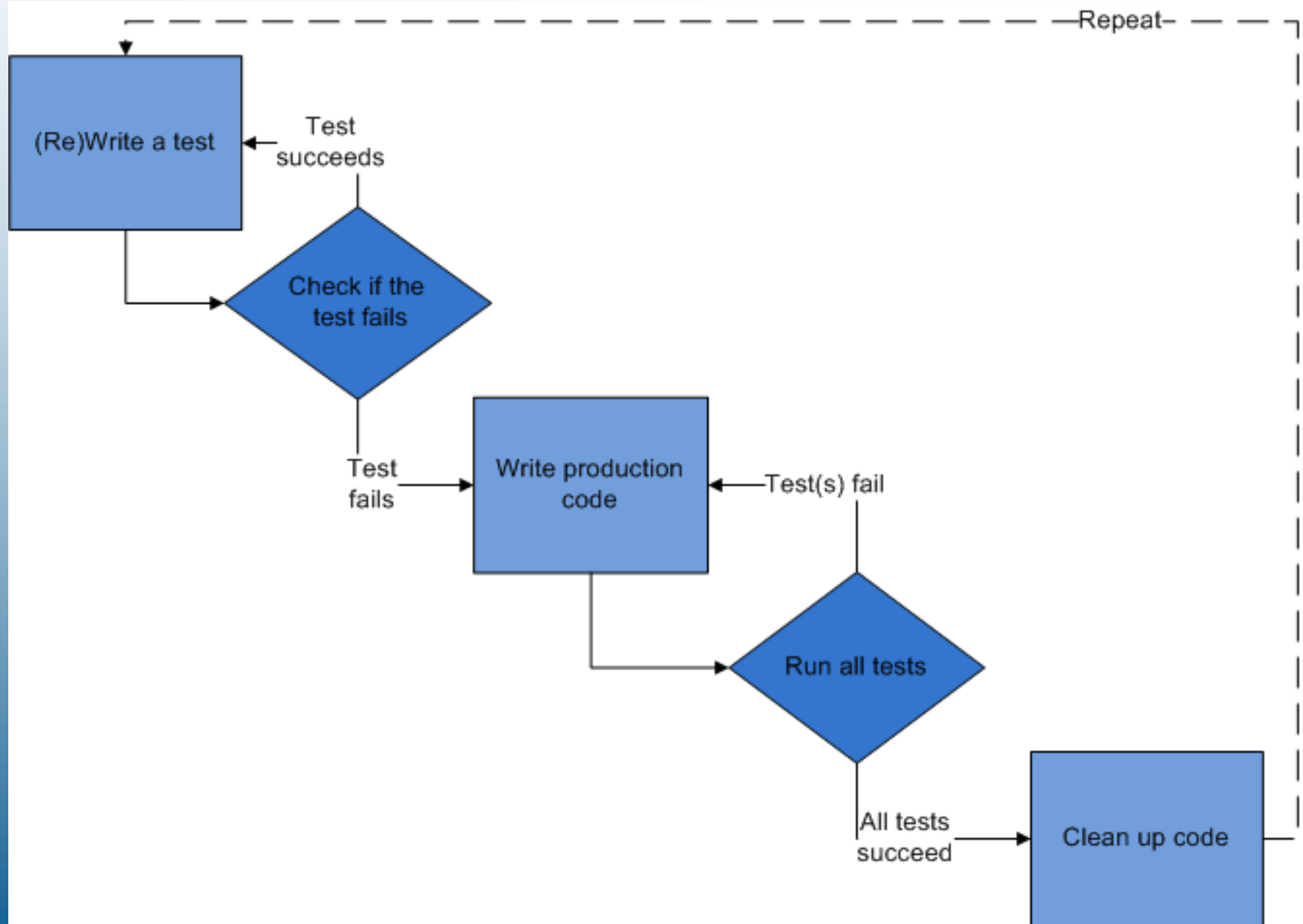
# TDD Test Driven Development

TDD refers to a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring).

It can be succinctly described by the following set of rules:

- Write a “single” unit test describing an aspect of the program
- Run the test, which should fail because the program lacks that feature
- Write “just enough” code, the simplest possible, to make the test pass
- Refactor the code until it conforms to the simplicity criteria
- Repeat, “accumulating” unit tests over time

# Test Driven Development





# Development methods - summary

- The most important are the people.
- Software development does not work without a methodology focused on:
  - Responsibilities
  - Requirements, Architecture and Uncertainties
  - Continuously revised answers to questions:
    - WHAT-HOW-WHERE-WHO-WHEN-WHY
  - Testing whether the implemented software meets the defined requirements.
- There must be a minimum project documentation.
- As a result, the only measure of the quality of project implementation is functional software with good quality of source code.



# Semestral project -Artifact 7


## Presentation your IS

Present your developed IS, which consists of:

- three implemented use cases
- two different graphical user interfaces
- two different data repositories
- complete documentation (one pdf A1-A6)

Presentation will be at the last exercise on 15.12.2021

# Exercise tasks

- Present your current state of semestral work code.
  - Discussion of the domain model focuses on the patterns used and the interaction of objects within the patterns.
  - Continue the implementation the semester project.
- 

# Lecture checking questions

- Describe what is meant by the waterfall model and what are its advantages and disadvantages.
- Describe what is meant by iterative and incremental development. Give an example.
- What is meant by UP (unified process), what are its principles, characteristics and phases? Give examples.
- What are the four basic characteristics of agile software development that distinguish it from waterfall and other robust approaches?
- What are the principles and practices underlying so-called extreme programming?
- What are the most important characteristics of SCRUM?
- Describe the process typical of so-called test-driven software development. Give an example.