

# Processes, Methodologies and Tools

---



# Who am I?

---

- ◆ Name: Maurizio Martignano
- ◆ Experience: working with Ada and C/C++ since 1983, with Java since 1996.
- ◆ Application domains: system software, embedded systems, satellites data management systems, radiation software, control automation.
- ◆ Currently he's Senior System Engineer at the European Space Agency and he's working on the following areas:
  - Embedded systems
  - Communication protocols
  - (Web based) Distributed Applications
  - XML based procedures
- ◆ Where: Noordwijk, The Netherlands (+31-71-5656749, Maurizio.Martignano@esa.int)

# Definitions

---

- ◆ **Process** = a definition of a set of activities (and not their execution)
- ◆ **Methodology** = a body of practices, procedures and rules on how to perform an activity
- ◆ **Tool** = a device, an instrument used to to implement, facilitate a task, an activity defined by a methodology

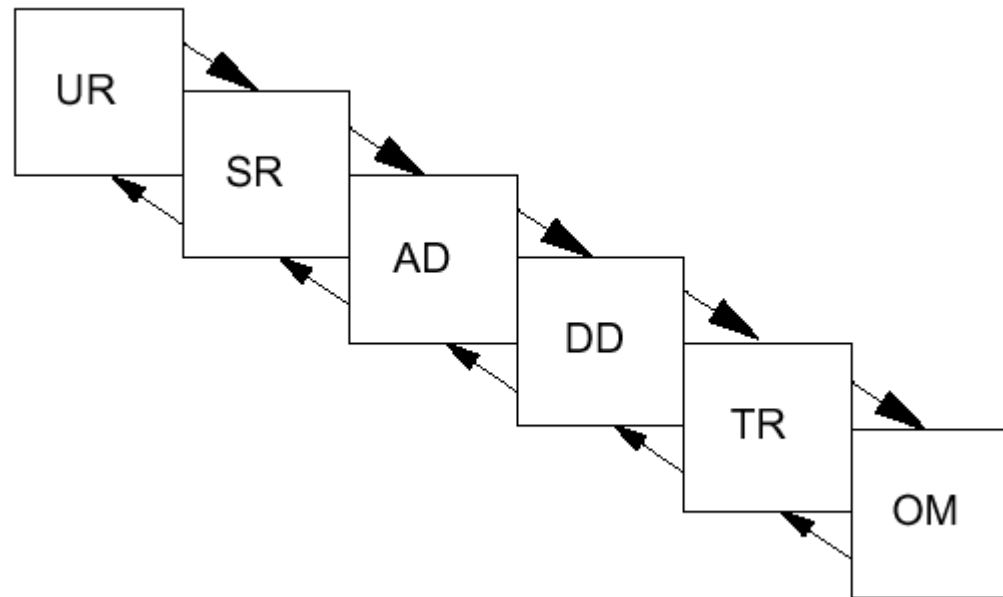
# Processes

---

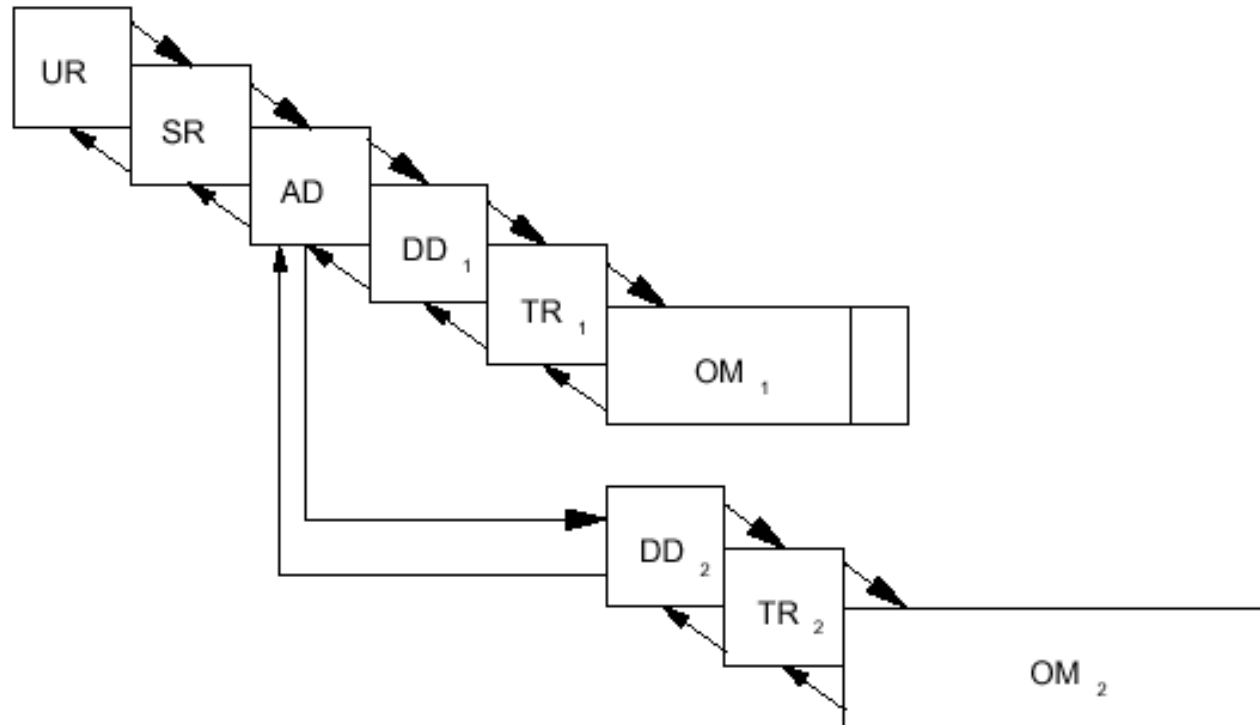
- ◆ Advantages of a common process
  - Everyone on the development team can understand what he has to do
  - Developers can better understand what other developers are doing
  - Supervisors and managers can understand what developers are doing
  - Supervisors, managers and developers can transfer between projects and or departments
  - Training can be standardised
  - The course of software development is repeatable

- ◆ Disadvantages of a commonly defined project
  - It can be too heavy...
  - Not flexible enough...
  - Not suitable for a particular project...
  
- ◆ It can be a process with an “upper case” P...

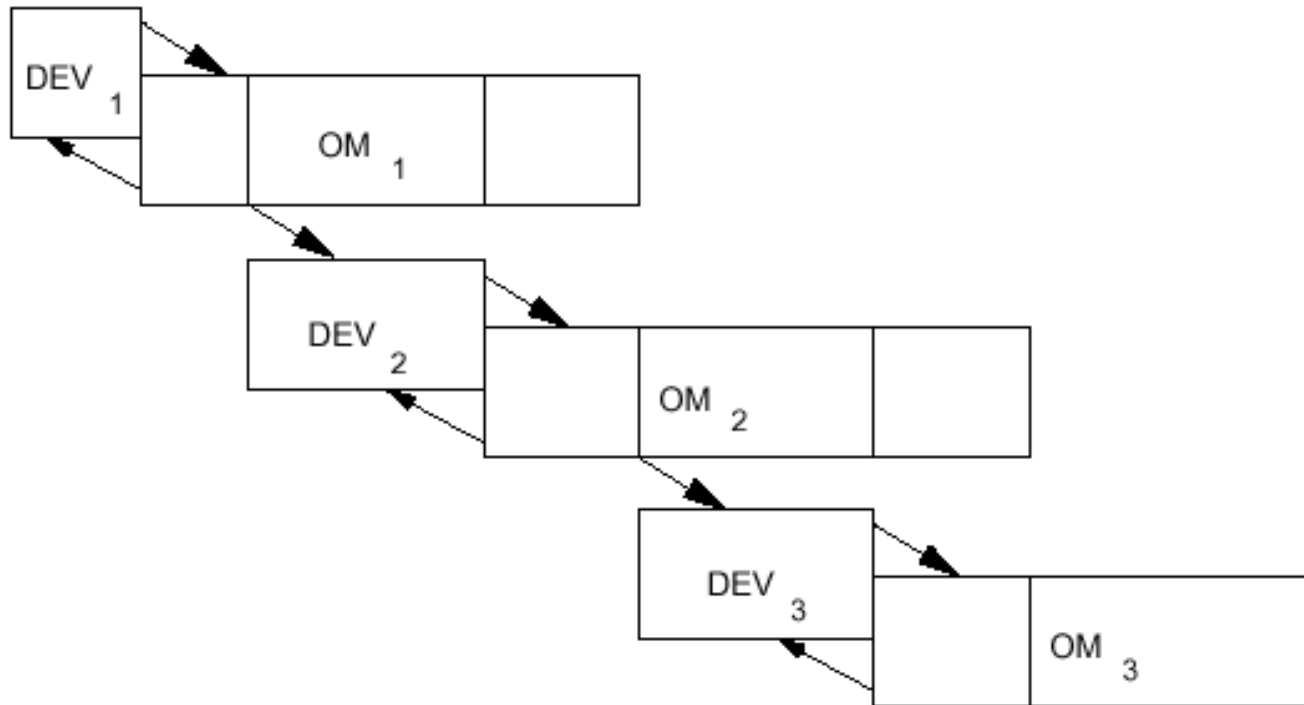
# Processes – Waterfall Model



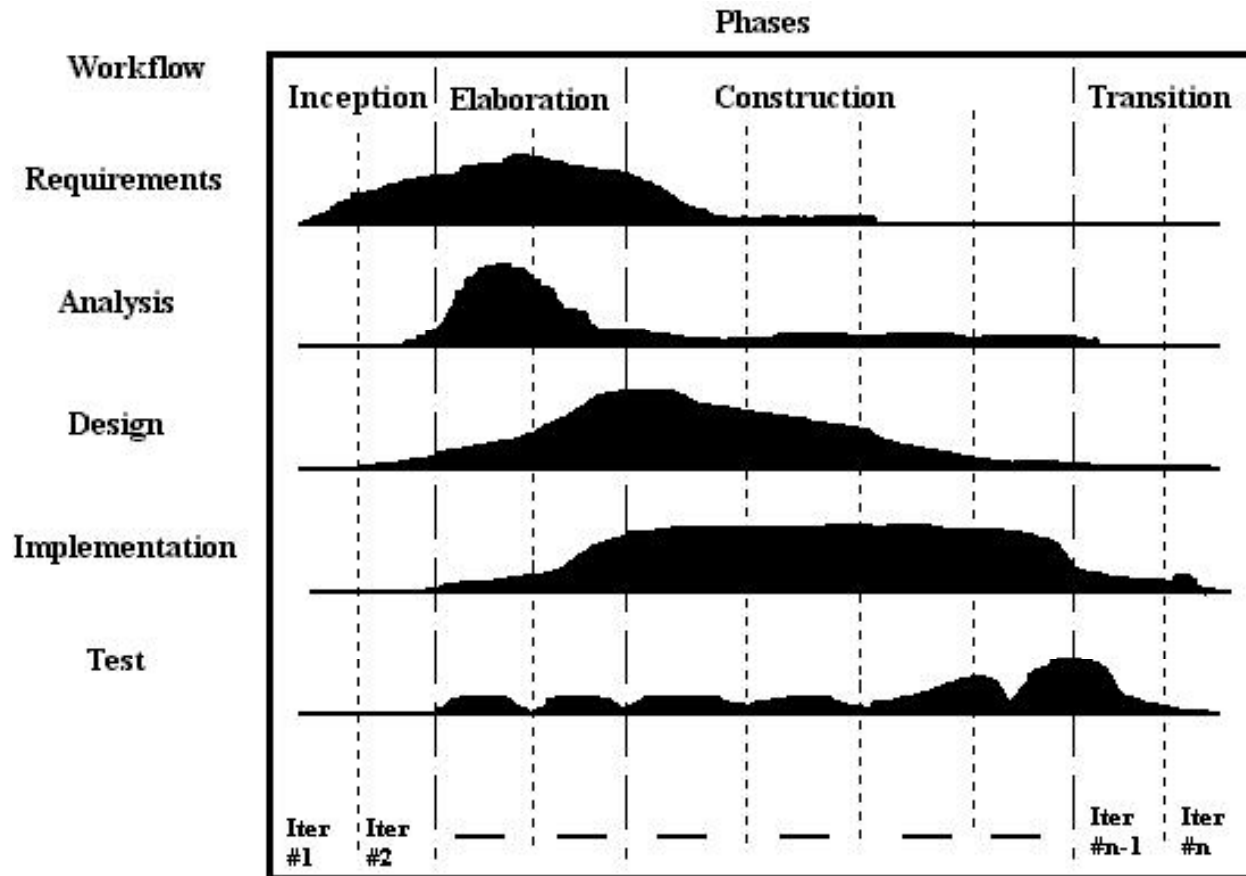
# Processes – Incremental Model



# Processes – Evolutionary Model



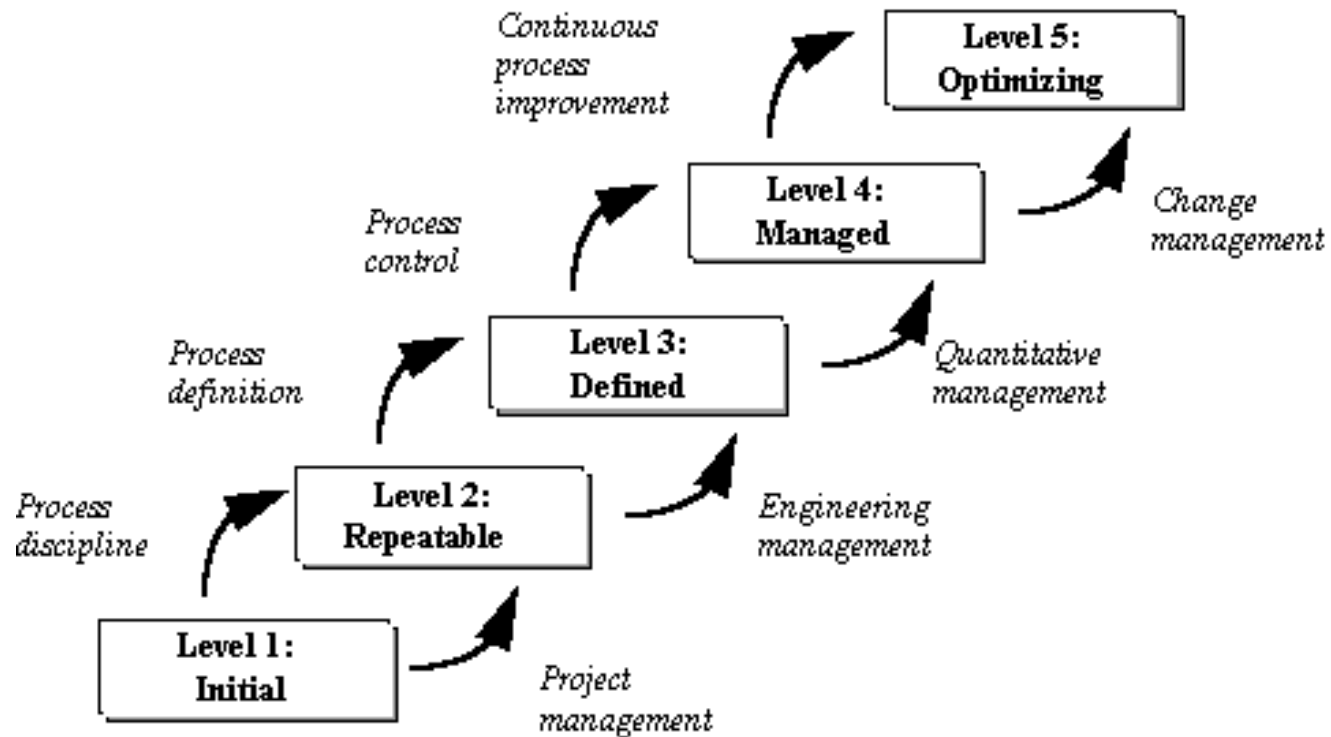
# Processes – Unified Software Development Model



# Processes: Vertical / Horizontal Activities

<b>Requirements</b>	<b>Analysis</b>	<b>Design</b>	<b>Implementation</b>	<b>Testing</b>
<b>Verification &amp; Validation</b>				
<b>Configuration Mng</b>				
<b>Quality</b>				
<b>Management</b>				

# Processes: Capability Maturity Model



# Methodologies (Only OOAD)

- ◆ The following is a list of some Object Oriented Analysis and Design Methods:

BON, Booch, BOOM, Catalysis, CBD/e, Coad/Yourdon, COMM A, CRC, Convergent Engineering, Demeter, DOORS, DOOS, EPA, EROOS, Fusion, Goofee, HOOD, IDEA, ION, KISS, MERODE, MOSES, MWOOD, Object COMX, Objecteering, Objectory, OEP, Octopus, OMT, OOAD/OOIE, OOA/RD, OOBE, OOCL, OOHDM, OOram, OOSC, OOSD, OOSE, OOSP, Open, OSA, PAUD, ROAD, ROPES, RUP, Scrum, Skill-Driven Design, SDL, Shlaer & Mellor, Softstar, SOMA, SOMT, Syntropy, XP

- ◆ Useful page:

[http://www.cetuslinks.org/oo\\_oaa\\_ood\\_methods.html](http://www.cetuslinks.org/oo_oaa_ood_methods.html)

# Methodologies

---

- ◆ In 1993 it was possible to identify the following schools of thought:
  - **Data Centered School** – Shlaer & Mellor, **Rumbauch's Objects Modeling Technique**
  - **Structural School** – Class-Responsibility-Collaboboration, **Booch**
  - **Scenario Based School** – Object Behaviour Analysis, **Jacobson's Object Oriented Software Engineering**

# UML History

---

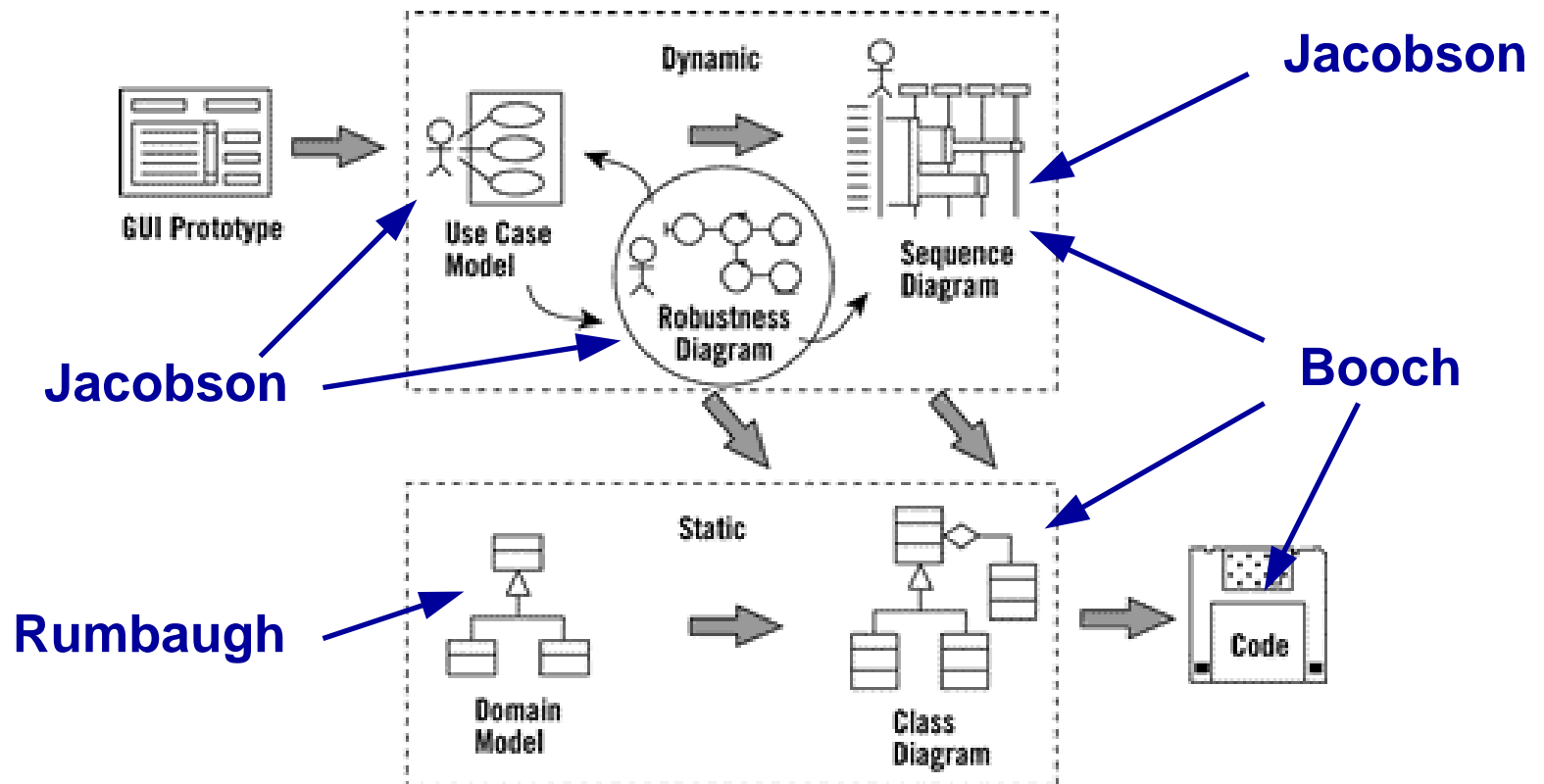
- ◆ Grady Booch and James Rumbaugh at Rational Software Corporation started the work on UML in 1994. Their goal was to create a new method, "Unified Method", that were to unite the Booch method and the OMT-2 method (of which Rumbaugh was the leading developer).
- ◆ In 1995, Ivar Jacobson joined, which is the man behind the OOSE method and the Objectory method.
- ◆ Rational also bought Objective Systems, the Swedish company that developed and distributed Objectory.
- ◆ At this point, the developers of UML also realized that their work was more aimed at creating a standard modeling language and renamed their work to "Unified Modeling Language".

# UML - History

---

- ◆ To succeed in establishing a standard for modeling language is also a much simpler task than doing the same for a process since a process differs substantially between different companies and cultures. It is doubtful if it is at all possible to create a standard process that can be used by everybody.
- ◆ Booch, Rumbaugh and Jacobson (often referred to as the "three amigos") released a number of preliminary versions of UML to the object-oriented community, which gave them many ideas and suggestions to improve the language. Version 1.0 of the Unified Modeling Language was released in January 1997.
- ◆ The latest version of UML is 1.3, released in June 1999.

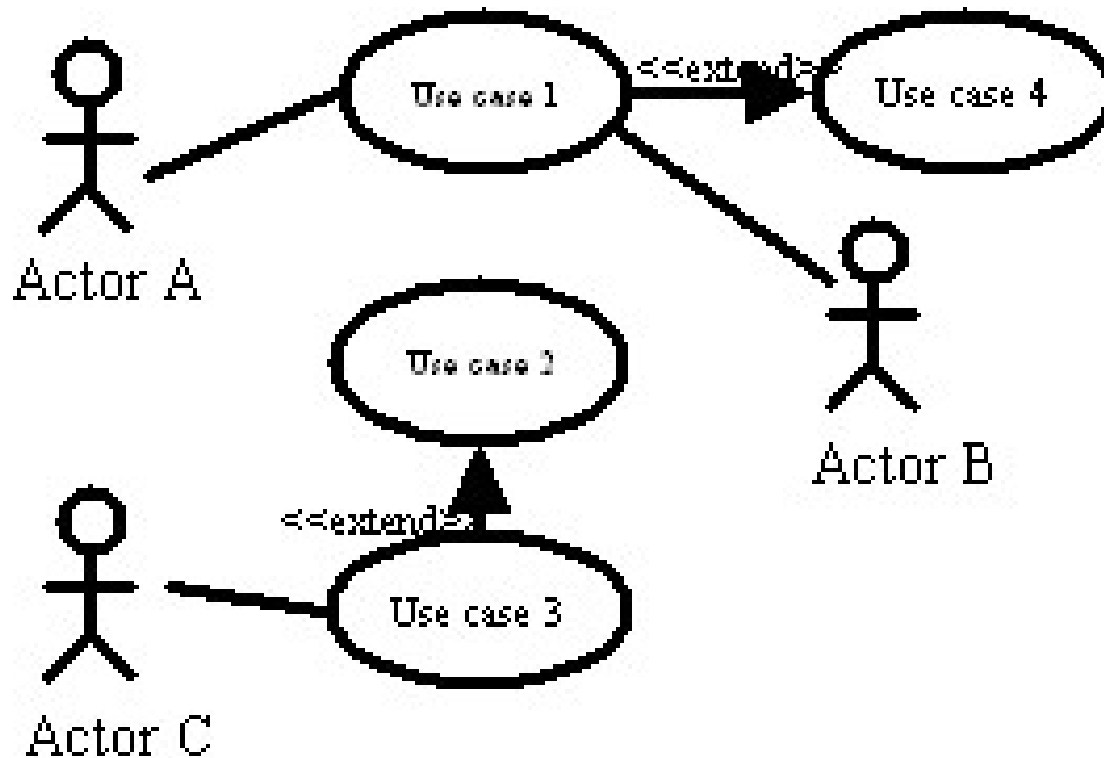
# UML - Contributions



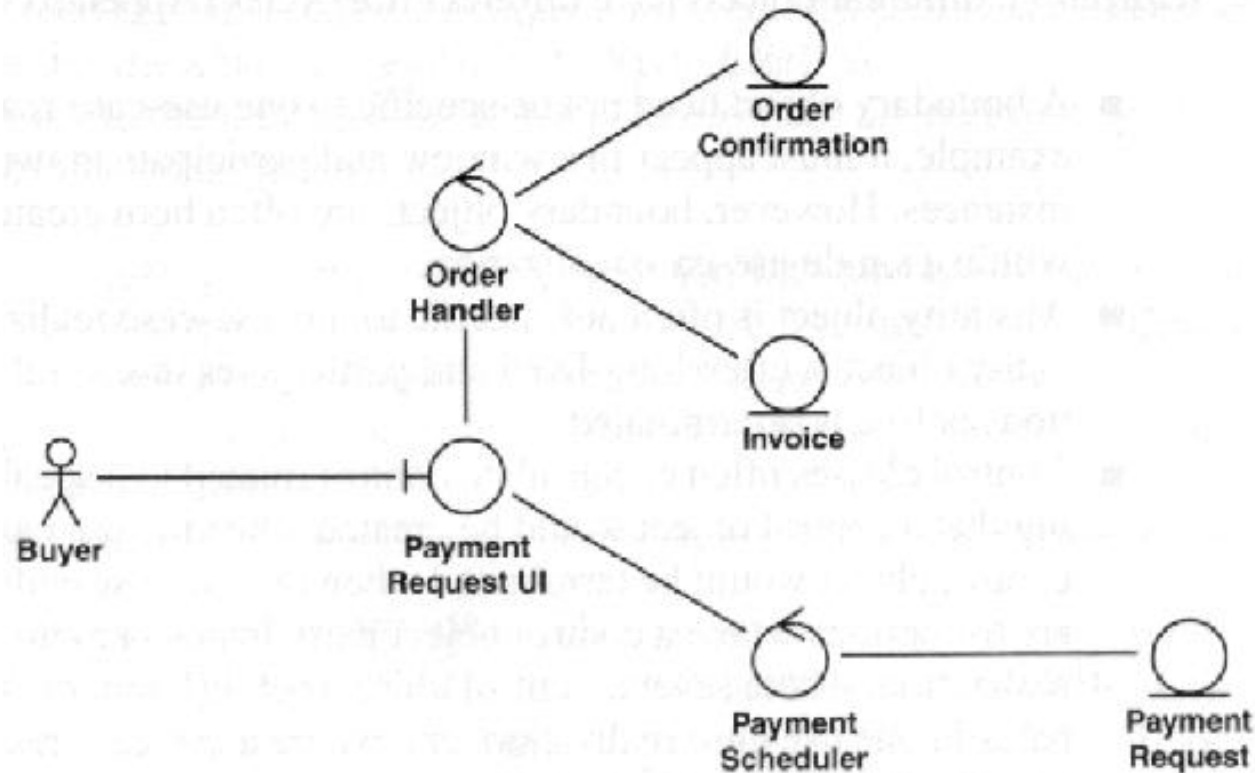
# UML Models according to the Unified Software Development Model

◆ Requirements	◆ Use-Case Model
◆ Analysis	◆ Analysis Model
◆ Design	◆ Design Model, Deployment Model
◆ Implementation	◆ Implementation Model
◆ Test	◆ Test Model

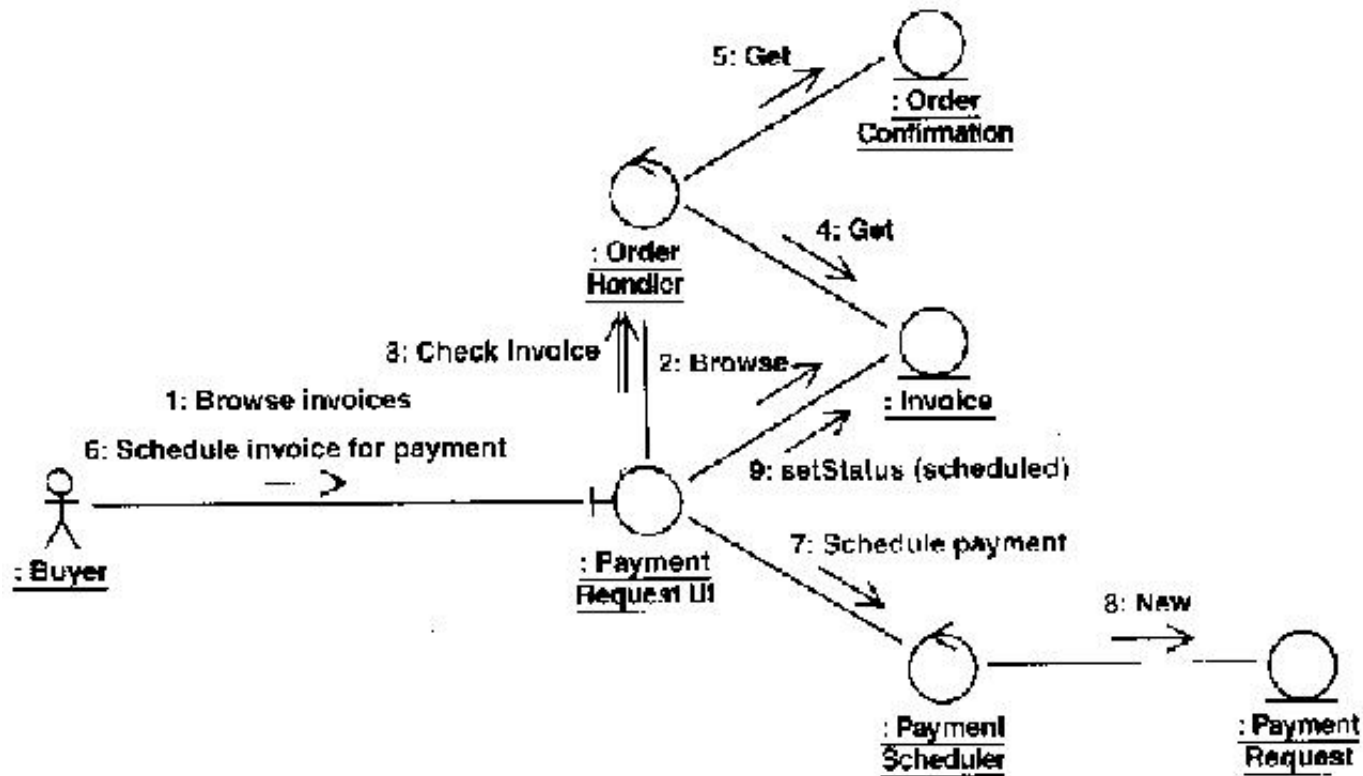
# Use-Case Model – Use-Case Diagram



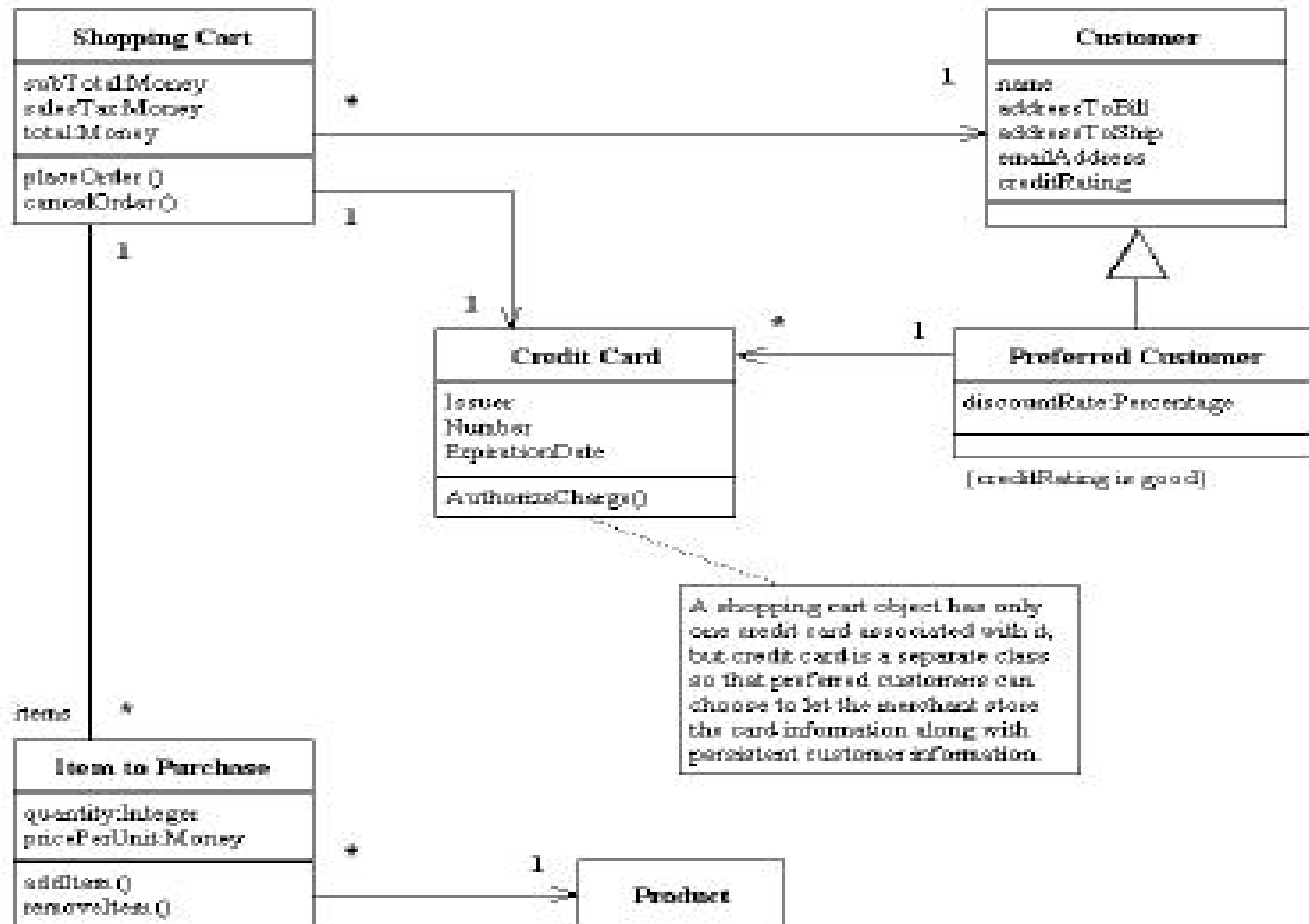
# Analysis Model – Class Diagram



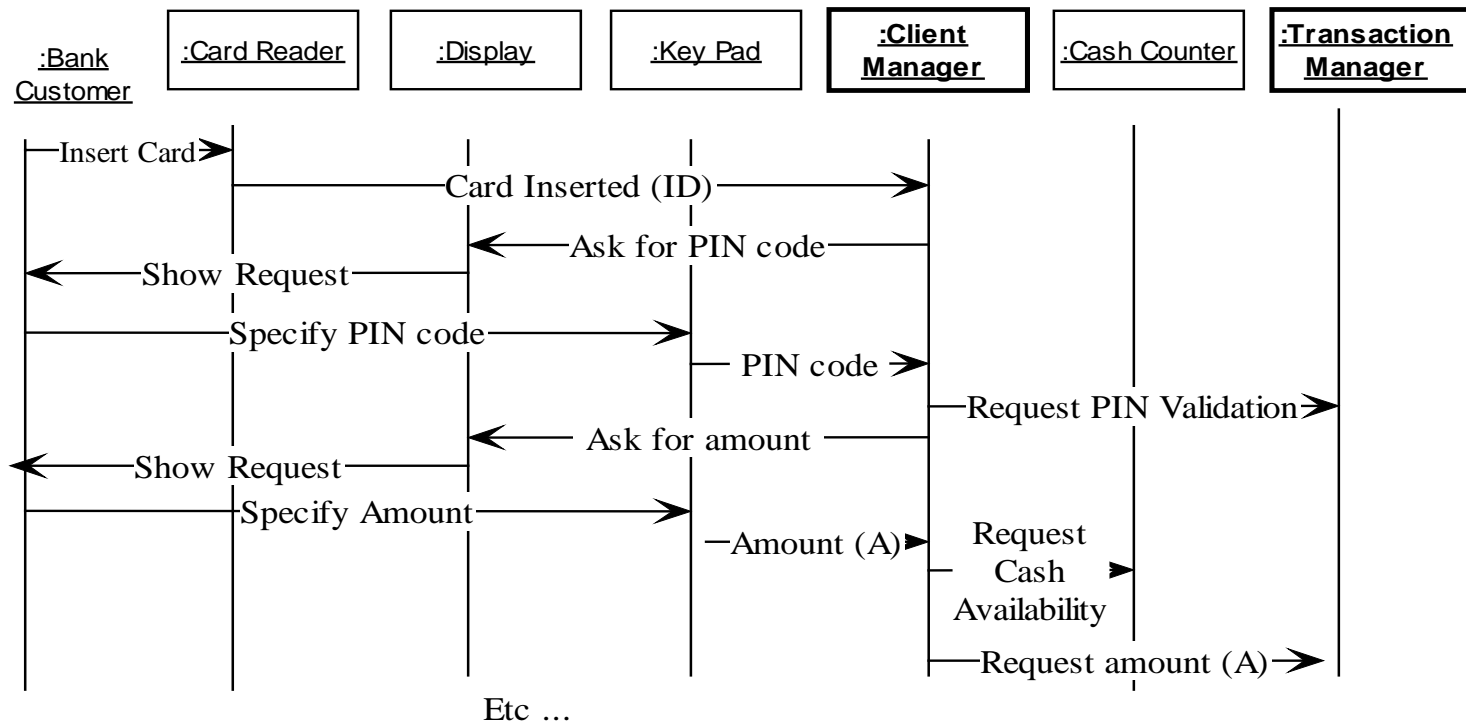
# Analysis / Design Models – Collaboration Diagram



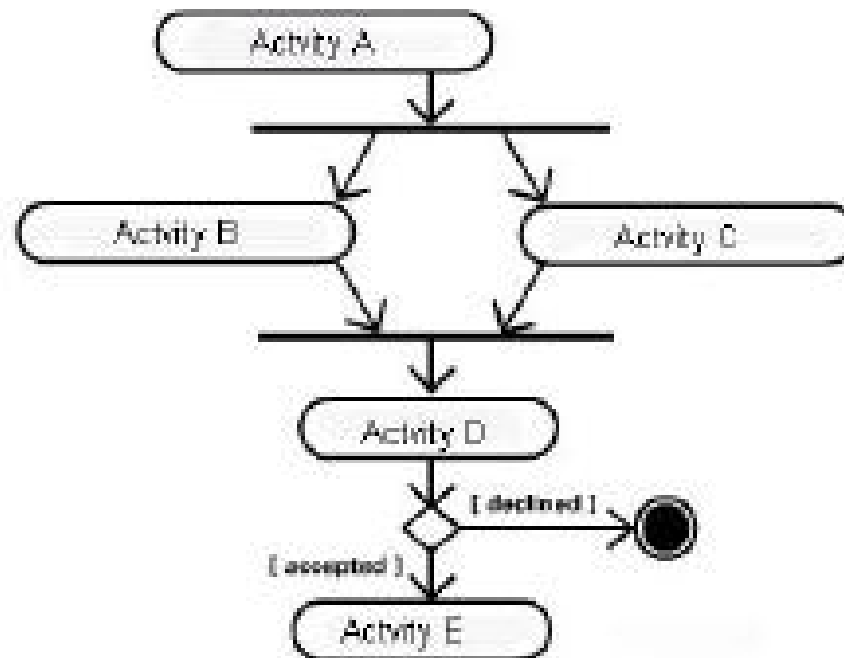
# Design Model – Class Diagram



# Design Model – Sequence Diagram



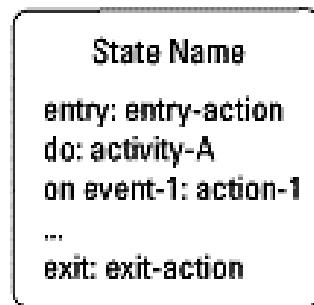
# Design Model – Activity Diagram



# Design Model – State Transition Diagram

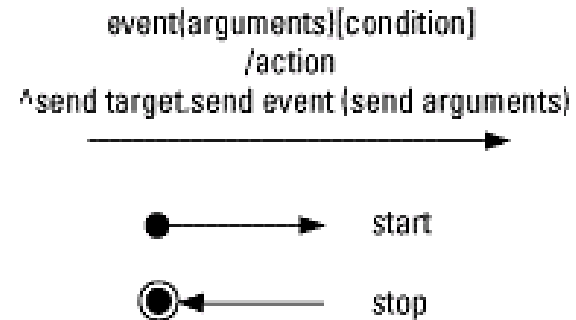
**STATE-TRANSITION DIAGRAM** Shows the state space of a given context, the events that cause a transition from one state to another, and the actions that result

## State icon

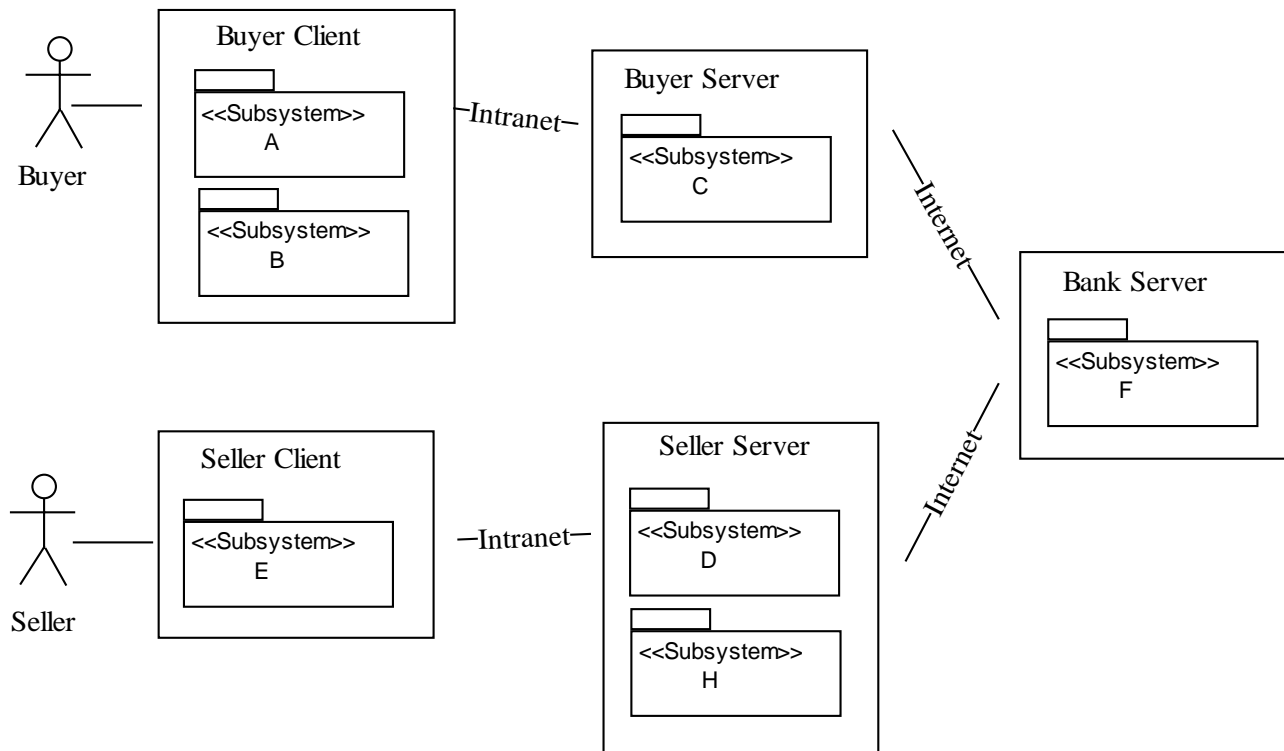


History (H)

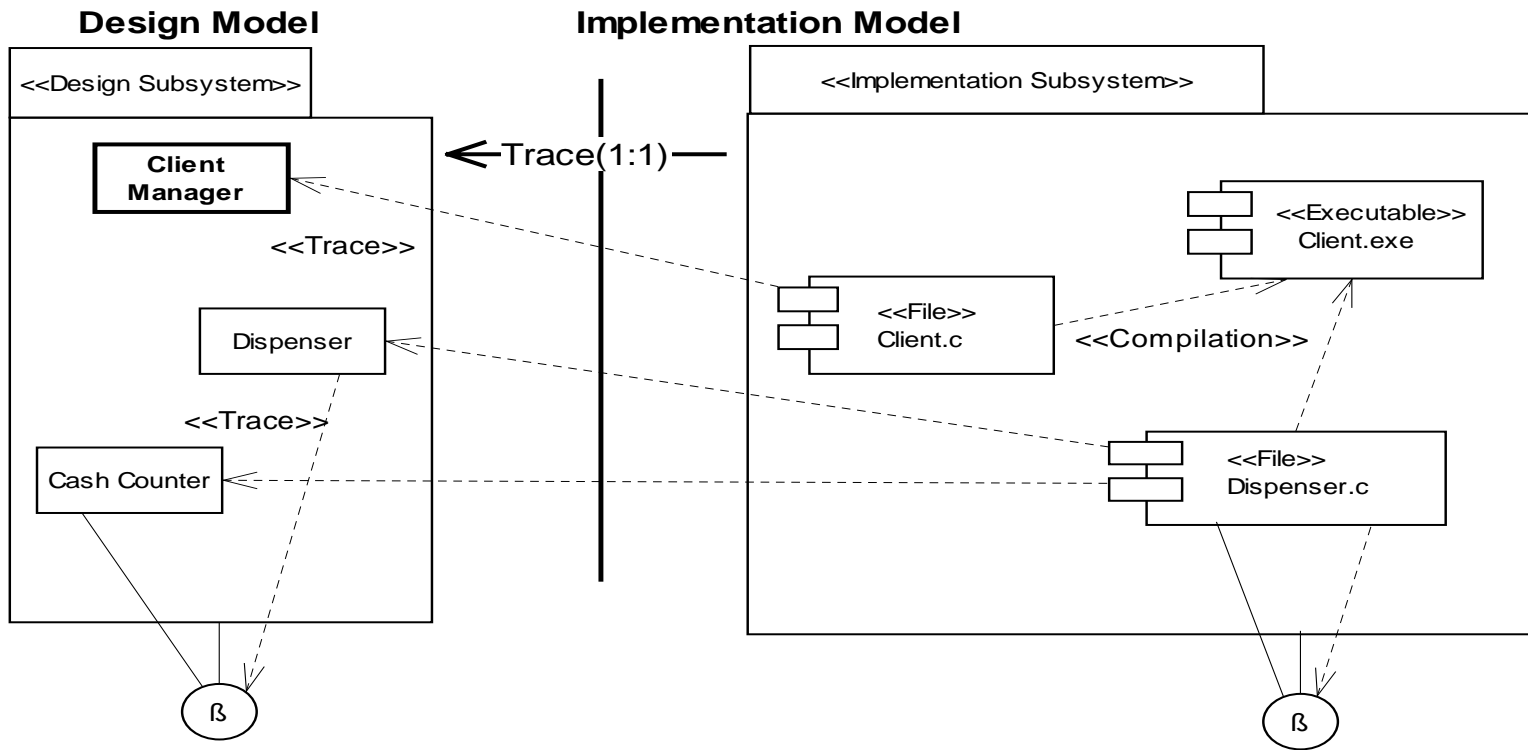
## State transitions



# Deployment Model – Deployment Diagram



# Implementation Model – Implementation Diagram



# Test Model

---

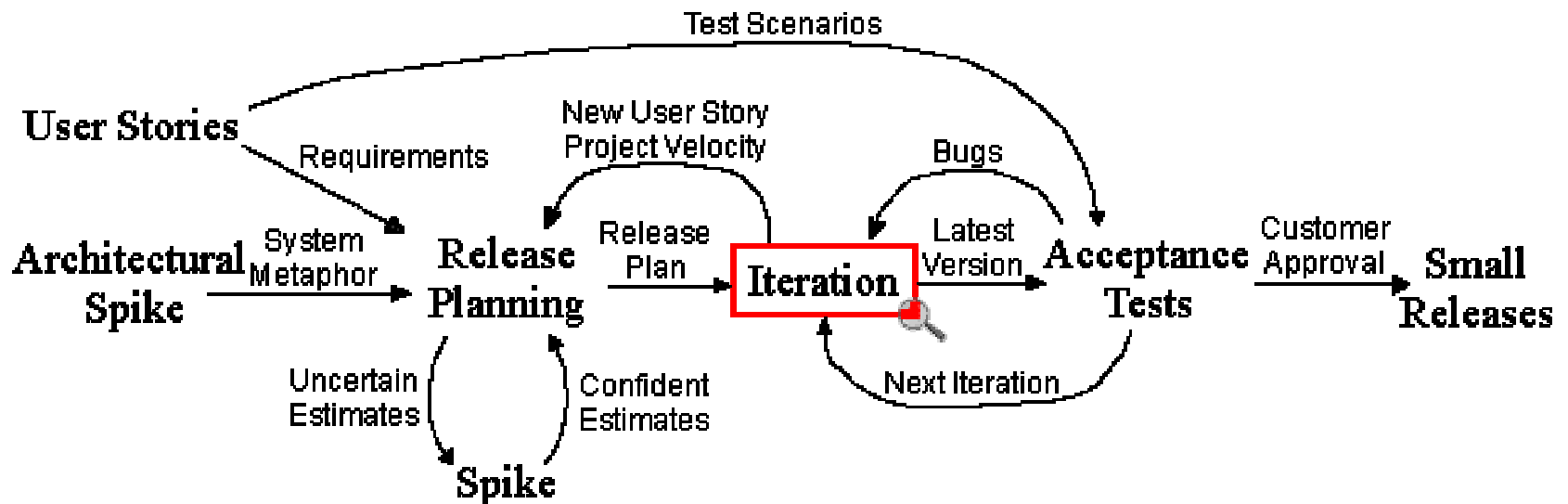
- ◆ Basically the test model is made of all the others models, whereas special attention is paid to testing, e.g.
  - Test use-cases
  - Test collaboration diagrams
  - Test components
  - Etc...

# eXtreme Programming

---

- ◆ Extreme Programming (XP) was created in response to problem domains whose requirements change.
- ◆ XP was also set up to address the problems of project risk.
- ◆ XP is set up for small groups of programmers. Between 2 and 10.
- ◆ XP requires an extended development team. The XP team includes not only the developers, but the managers and customers as well, all working together elbow to elbow.
- ◆ Another requirement is testability. You must be able to create automated unit and functional tests.
- ◆ The last thing on the list is productivity. XP projects unanimously report greater programmer productivity.
- ◆ Kent Beck, Ward Cunningham

# eXtreme Programming - Process



# XP – User Stories

---

- ◆ User stories serve the same purpose as use cases but are not the same.
- ◆ They are used to create time estimates for the release planning meeting. They are also used instead of a large requirements document.
- ◆ User Stories are written by the customers as things that the system needs to do for them. They are similar to usage scenarios, except that they are not limited to describing a user interface.
- ◆ They are in the format of about three sentences of text written by the customer in the customers terminology without techno-syntax.
- ◆ User stories also drive the creation of the acceptance tests. One or more automated acceptance tests must be created to verify the user story has been correctly implemented.

# XP – (Architectural) Spike

---

- ◆ Create spike solutions to figure out answers to tough technical or design problems.
- ◆ A spike solution is a very simple program to explore potential solutions.
- ◆ Build a system which only addresses the problem under examination and ignore all other concerns.
- ◆ Most spikes are not good enough to keep, so expect to throw it away.
- ◆ The goal is reducing the risk of a technical problem or increase the reliability of a user story's estimate.

# XP – Release Planning

- ◆ A release planning meeting is used to create a release plan, which lays out the overall project.
- ◆ The release plan is then used to create iteration plans for each individual iteration.
- ◆ It is important for technical people to make the technical decisions and business people to make the business decisions.
- ◆ The essence of the release planning meeting is for the development team to estimate each user story in terms of ideal programming weeks.
- ◆ User stories are printed or written on cards. Together developers and customers move the cards around on a large table to create a set of stories to be implemented as the first (or next) release. A useable, testable system that makes good business sense delivered early is desired.

# XP – Release Planning

(cont)

- ◆ The project velocity (or just velocity) is a measure of how fast work is getting done on your project.
- ◆ To measure the project velocity you simply count up how many user stories, or how many programming tasks were finished during the iteration

# XP - Iteration

---

- ◆ Iterative Development adds agility to the development process.
- ◆ Divide your development schedule into about a dozen iterations of 1 to 3 weeks in length.
- ◆ Don't schedule your programming tasks in advance. Instead have an iteration planning meeting at the beginning of each iteration to plan out what will be done.
- ◆ It is also against the rules to look ahead and try to implement anything that it is not scheduled for this iteration.
- ◆ There will be plenty of time to implement that functionality when it becomes the most important story in the release plan.

# XP - Development

---

- ◆ Pair Programming
- ◆ Refactor Mercilessly
- ◆ Move People Around
- ◆ Test & Test & Test

# XP – Acceptance Test

---

- ◆ Acceptance tests are created from user stories. During an iteration the user stories selected during the iteration planning meeting will be translated into acceptance tests.
- ◆ The customer specifies scenarios to test when a user story has been correctly implemented. A story can have one or many acceptance tests, what ever it takes to ensure the functionality works.
- ◆ Acceptance tests are black box system tests. Each acceptance test represents some expected result from the system.
- ◆ Customers are responsible for verifying the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of highest priority.
- ◆ A user story is not considered complete until it has passed its acceptance tests.

# XP – Small Releases

---

- ◆ The development team needs to release iterative versions of the system to the customers often.
- ◆ The release planning meeting is used to discover small units of functionality that make good business sense and can be released into the customer's environment early in the project.
- ◆ This is critical to getting valuable feedback in time to have an impact on the system's development.
- ◆ The longer you wait to introduce an important feature to the system's users the less time you will have to fix it.

# XP - Rules

---

## ◆ Planning

- User stories are written.
- Release planning creates the schedule.
- Make frequent small releases.
- The Project Velocity is measured.
- The project is divided into iterations.
- Iteration planning starts each iteration.
- Move people around.
- A stand-up meeting starts each day.
- Fix XP when it breaks.

## ◆ Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible

## ◆ Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Use collective code ownership.
- Leave optimization till last.
- No overtime.

## ◆ Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

# Use-Case Driven Object Modeling with UML

---

- ◆ Since 1993 Doug Rosenber and Kendall Scott have been working at ICONIX(?)-ICONICS on the definition of a method merging together some parts of OMT, OOSE and Booch
- ◆ As soon as UML appeared they used the new language as foundation for their method.
- ◆ This approach has been developed in ICONIX since 1993 and it was based on a mixture of OMT, OOSE and Booch.

# UCDOM – Requirement Analysis

---

- ◆ Identify your real-world domain objects and the generalisation and aggregation relationships among those objects. Start drawing a high-level class diagram.
- ◆ If it's feasible, do some rapid prototyping of the proposed system. Or gather whatever substantive information you have about the legacy system you are reengineering.
- ◆ Identify your use cases, using use case diagrams.
- ◆ Organise the use cases into groups. Capture this organisation in a package diagram.
- ◆ Allocate functional requirements to the use cases and domain objects at this stage.
- ◆ **Milestone 1: Requirements Review**

# UCDOM – Preliminary Design

---

- ◆ Write descriptions of the use cases – basic courses of actions that represent the “mainstream” and alternative courses for less-frequently traveled paths and error conditions.
- ◆ Perform robustness analysis. For each use case:
  - Identify a first cut of objects that accomplish the stated scenario. Use the UML Objectory stereotypes (Actor, Interface, Process, Entity)
  - Update your domain model class diagram with new objects and attributes as you discover them.
- ◆ Finish updating the class diagram so that it reflects the completion of the analysis phase of the project.
- ◆ **Milestone 2: Preliminary Design Review**

# UCDOM - Design

---

- ◆ Allocate behaviour, for each use case:
  - Identify the messages that need to be passed between objects, the objects and the associated methods to be invoked.
  - Draw a sequence diagram with use case text running down the left side and design information on the right.
  - Continue to update the class diagram with attributes and operations as you find them.
  - If you wish, use a collaboration diagram to show the key transactions between objects.
  - If you wish use a state diagram to show real-time behaviour.

- ◆ Finish the static model by adding detailed design information (for instance, visibility values and patterns).
- ◆ Verify with your team that your design satisfies all the requirements you've identified.
- ◆ **Milestone 3: Detailed / Critical Design Review**

# UCDOM - Implementation

---

- ◆ As needed, produce diagrams, such as deployment and component diagrams, that will help you with the implementation phase.
- ◆ Write / generate the code.
- ◆ Perform unit and integration testing.
- ◆ Perform system and user acceptance testing, using the use cases as black-box test cases for the latter.
- ◆ **Milestone 4: Delivery**

# Conclusions

---



**Your comments, please...**