# Software Engineering with Objects and Components

Massimo Felici

Room 1402, JCMB, KB

0131 650 5899

mfelici@inf.ed.ac.uk

# Administration

- SEOC webpage

  http://www.inf.ed.ac.uk/teaching/courses/seoc/

- Course Resources
  - Lecture Notes and References
  - Software: ArgoUML, Eclipse and Java
- Tutorials begin in week 3;
  - Frequency: once a week
  - Maximum 12 people per tutorial group
- Coursework:
  - in small teams (approx 3-4 people)
  - two deliverables equally weighted
    - 1st deliverable: Monday, 30th October
    - 2nd deliverable: Monday, 4th December
- Assessment:
  - 25% coursework; 75% degree examination

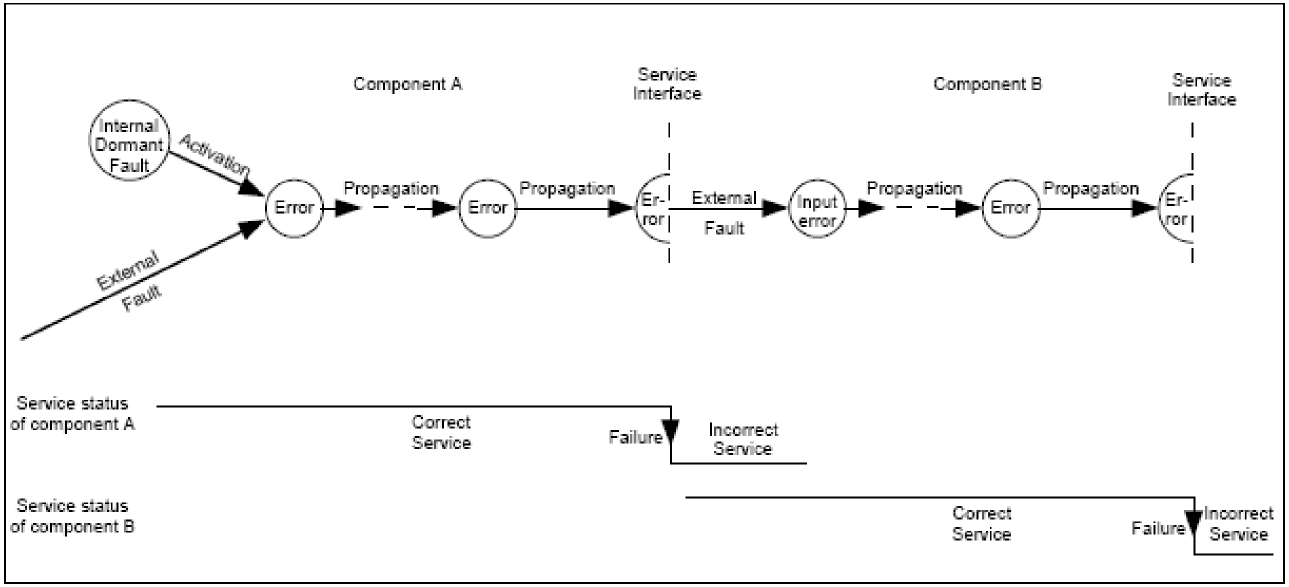
# Software Engineering

- *Software Engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.* [Sommerville 2007]
- *The right software delivered defect free, on time and on cost, every time.* [Software Engineering Institute (SEI)]
- Software Engineering studies
  - How to make software that is **“fit for purpose”**
    - good enough (functionally, non-functionally), meets constraints of the environment, law, ethics and work practice
  - How to meet **time** and **financial constraints** on delivery
- We still **fail** too often

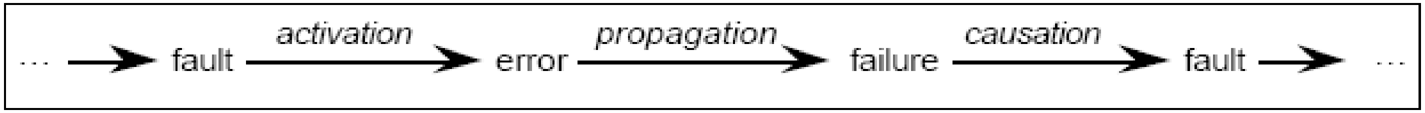
# Faults, Errors and Failures

- How does **Software Engineering** relate to **Faults**, **Errors** and **Failures**?
  - Design faults, Errors trigger failures, Software (system) fails, etc.
- Some **Definitions**…
  - **Failure** is the nonperformance or inability of the system or component to perform its intended function for a specified time under specified environmental conditions [Levenson 1995]
  - **Error** is a design flaw or deviation from a desired or intended state [Levenson 1995]
  - **Failure** is an event that occurs when the delivered service deviates from correct service [Avižienis et al.]
  - A **fault** is active when it causes an error [Avižienis et al.]
  - An **error** is the part of the total state of the system that may lead to its subsequent service failure. [Avižienis et al.]
- Warnings: slightly different use of faults, errors and failures

# The Pathology of Failure



- Relationship between **Faults**, **Errors** and **Failures**



- The fundamental chain of dependability threats

# An Example: Patriot Missile

- **Accident Scenario**: On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people.
- A report of the General Accounting office, GAO/IMTEC-92-26, entitled *Patriot Missile Defense: **Software Problem Led to System Failure** at Dhahran, Saudi Arabia* reported on the cause of the failure.

# Patriot Missile continued…

- **Fault**: inaccurate calculation of the time since boot due to computer arithmetic errors.
  - The time in tenths of second as measured by the system's internal clock was multiplied by 1/10 to produce the time in seconds.
  - This calculation was performed using a 24 bit fixed point register. In particular, the value 1/10, which has a non-terminating binary expansion, was chopped at 24 bits after the radix point.
- **Error**: The small chopping error, when multiplied by the large number giving the time in tenths of a second, lead to a significant error. Indeed, the Patriot battery had been up around 100 hours, and an easy calculation shows that the resulting time error due to the magnified chopping error was about **0.34 seconds**.
  - the binary expansion of 1/10 is

    **0.0001100110011001100110011001100....**
  - Now the 24 bit register in the Patriot stored instead

    **0.00011001100110011001100** introducing an error of

    **0.0000000000000000000000011001100...** binary, or about **0.000000095** decimal.
  - Multiplying by the number of tenths of a second in 100 hours gives 0.000000095×100×60×60×10=0.34.

# Patriot Missile continued…

- **Failure**: A Scud travels at about 1,676 meters per second, and so travels more than 500 meters in this time.
- This was far enough that the incoming Scud was outside the **"range gate"** that the Patriot tracked.
- Ironically, the fact that the bad time calculation had been improved in some parts of the code, but not all, contributed to the problem, since it meant that the inaccuracies did not cancel.

Figure 4: Calculated Range Gate After Approximately 8 Hours

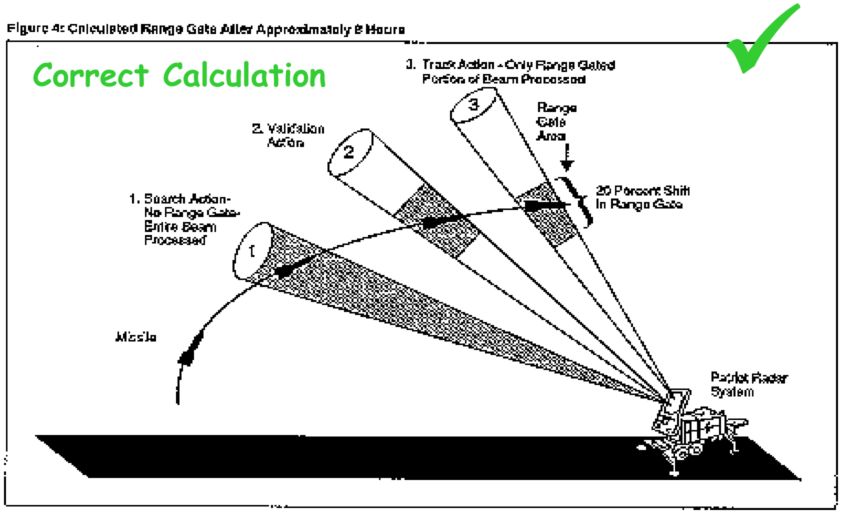
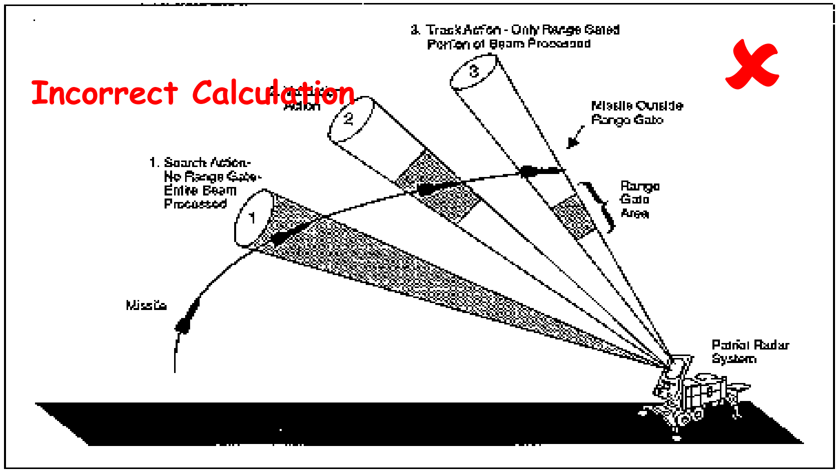

Correct Calculation ✓
3. Track Action - Only Range Gated Portion of Beam Processed
2. Validation Action
Range Gate Area
20 Percent Shift In Range Gate
1. Search Action- No Range Gate- Entire Beam Processed
1
2
3
Missile
Patriot Radar System

Figure 5: Incorrectly Calculated Range Gate

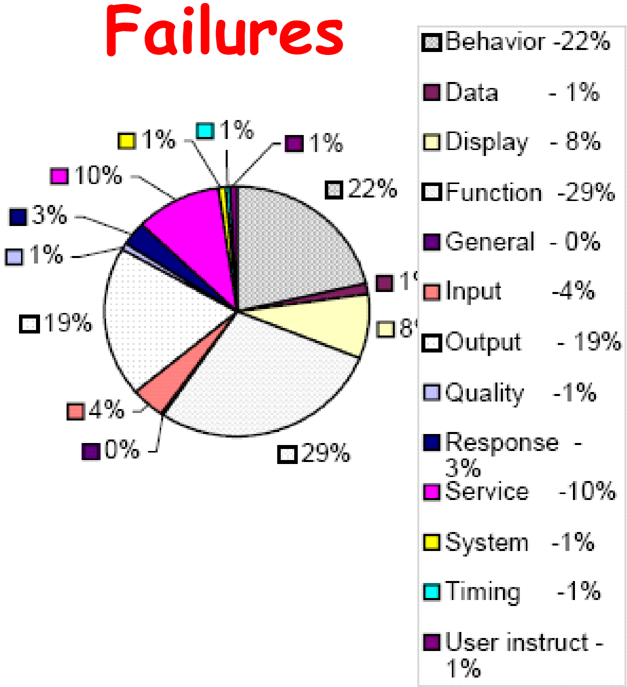

Incorrect Calculation ✗
3. Track Action - Only Range Gated Portion of Beam Processed
2. Validation Action
Missile Outside Range Gate
1. Search Action- No Range Gate- Entire Beam Processed
Range Gate Area
1
2
3
Missile
Patriot Radar System

# Patriot Missile …conclusions

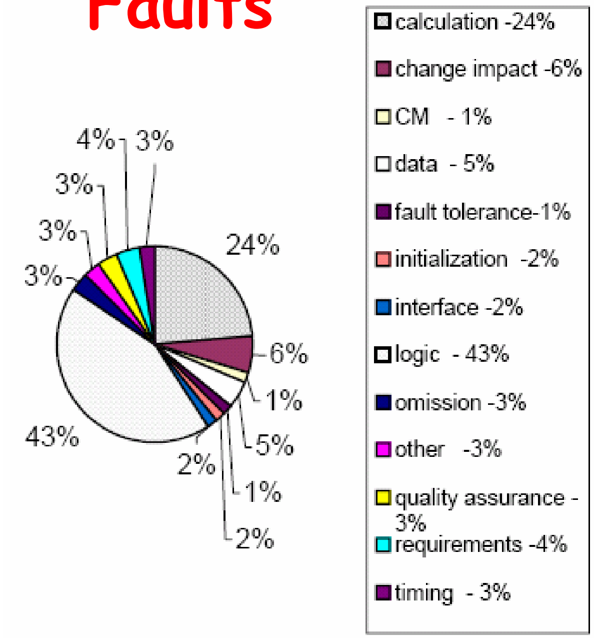
- Containing coding errors is very hard
  - seemingly insignificant errors result in major changes in behaviour
- Original fix suggested a change in procedures
  - reboot every 30 hours – impractical in operation
- Patriot is atypical
  - coding bugs rarely cause accidents alone
- Maintenance failure
  - failure of coding standards and traceability

# Other lessons from (medical device) failures

## Failures



## Faults



## Prevention

- Design, traceability, change impact analysis, configuration management, input data validation, fault tolerance, etc.

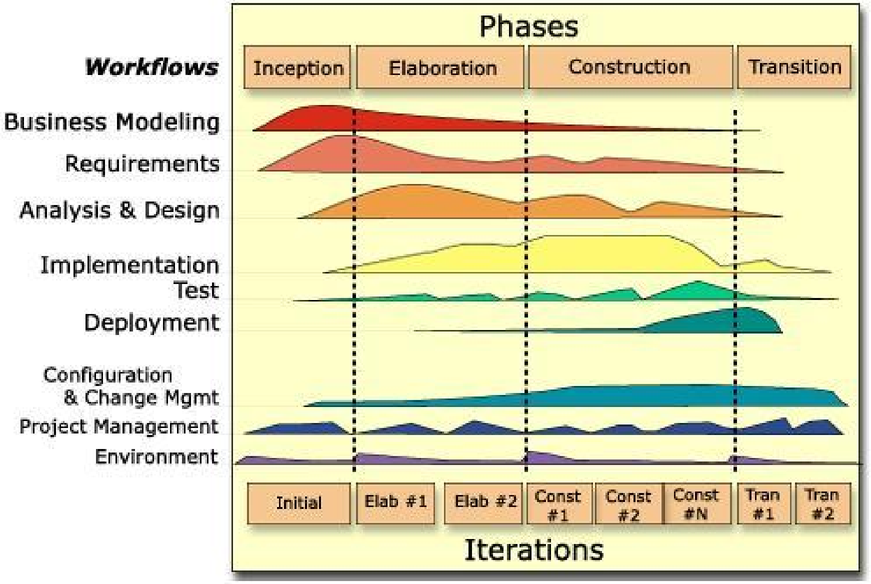
## Detection

- (Code) inspections, (unit or integration) testing, etc.

# Software Engineering

- We will study the following areas:
  - **Software Requirements**: the activities involved in gaining an accurate idea of what the users of the system want it to do.
  - **Software Design**: the design of a system to meet the requirements.
  - **Software Construction**: the realisation of the design as a program.
  - **Software Testing**: the process of checking the code meets the design.
  - **Software Configuration, Operation and Maintenance**: major cost in the lifetime of systems.
- These are the essential activities
- How we deploy effort and arrange these activities is part of Software Engineering Processes

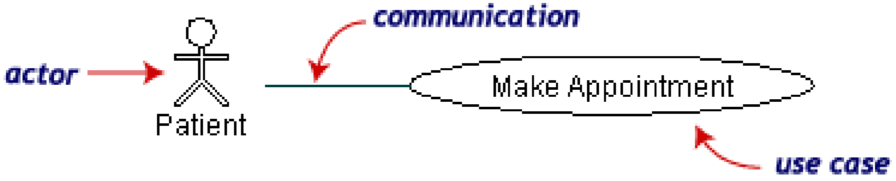
## (Rational) Unified Process - RUP



# Models Supporting SE

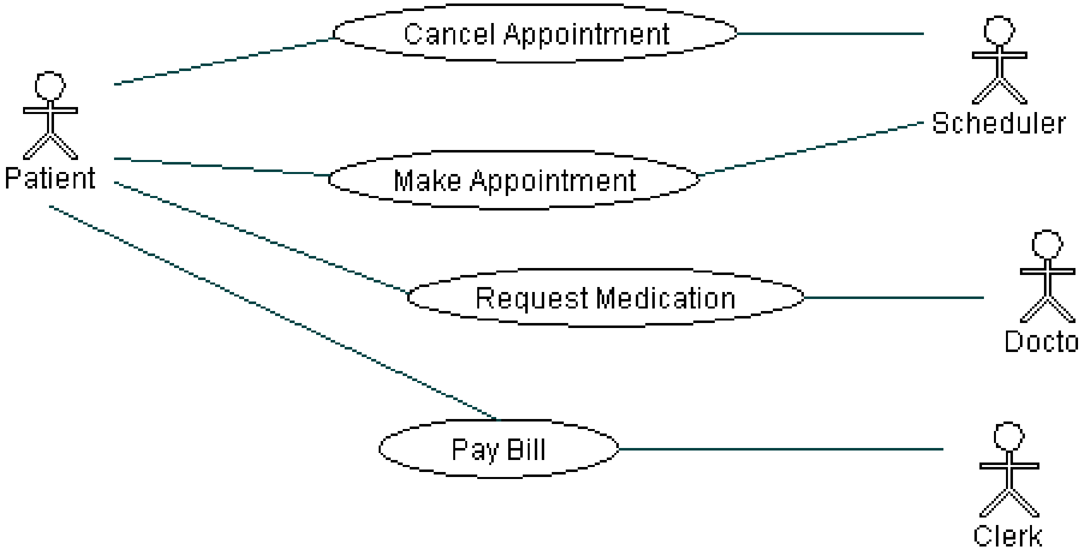
- UML provides a range of **graphical notations** that capture various aspects of the engineering process
- Provides a common notation for various different facets of systems
- Provides the basis for a range of consistency checks, validation and verification procedures
- Provides a common set of languages and notations that are the basis for creating tools

# UML: Use Case Diagrams



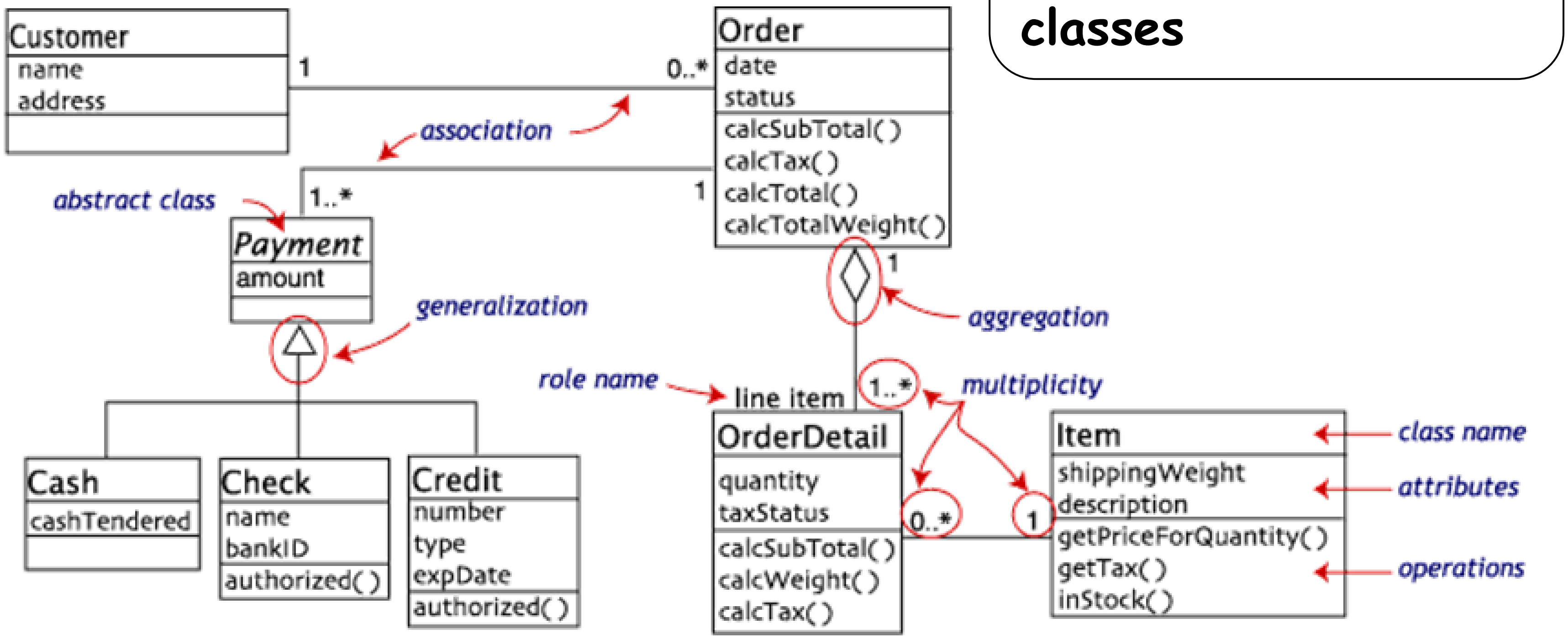
Used to support requirements Capture and analysis

Show the actors' Involvement in System activities

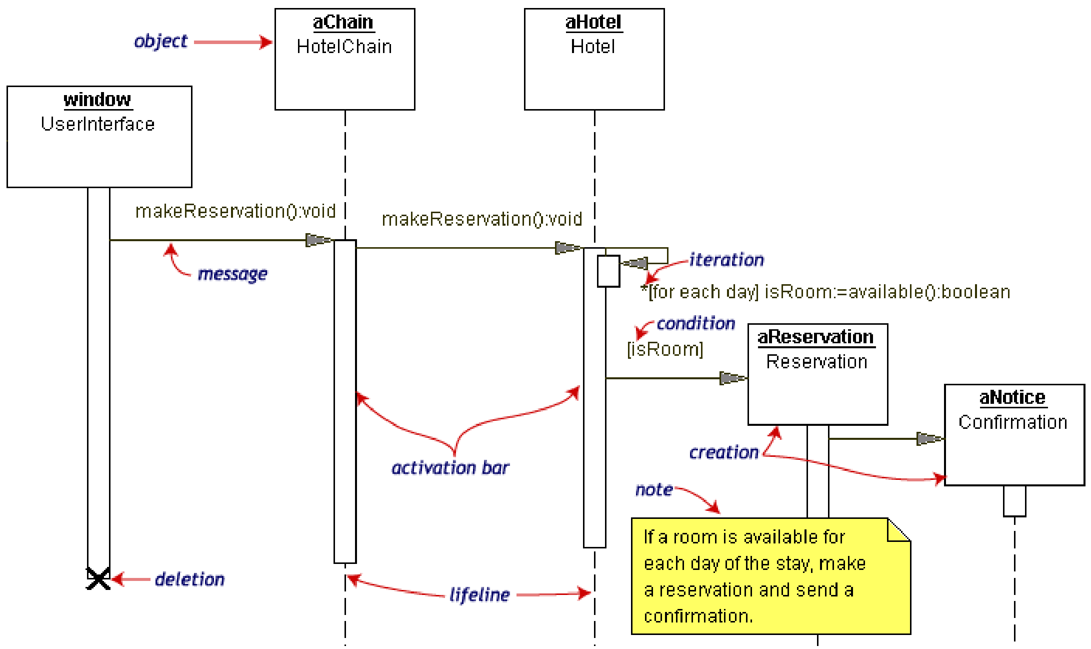


# UML: Class Diagrams

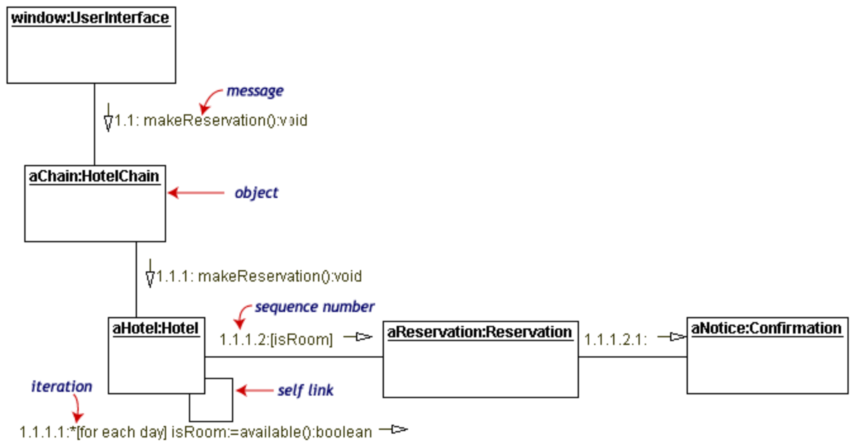
Capture the static structure of systems associations between classes



# UML: Sequence and Collaboration Diagrams

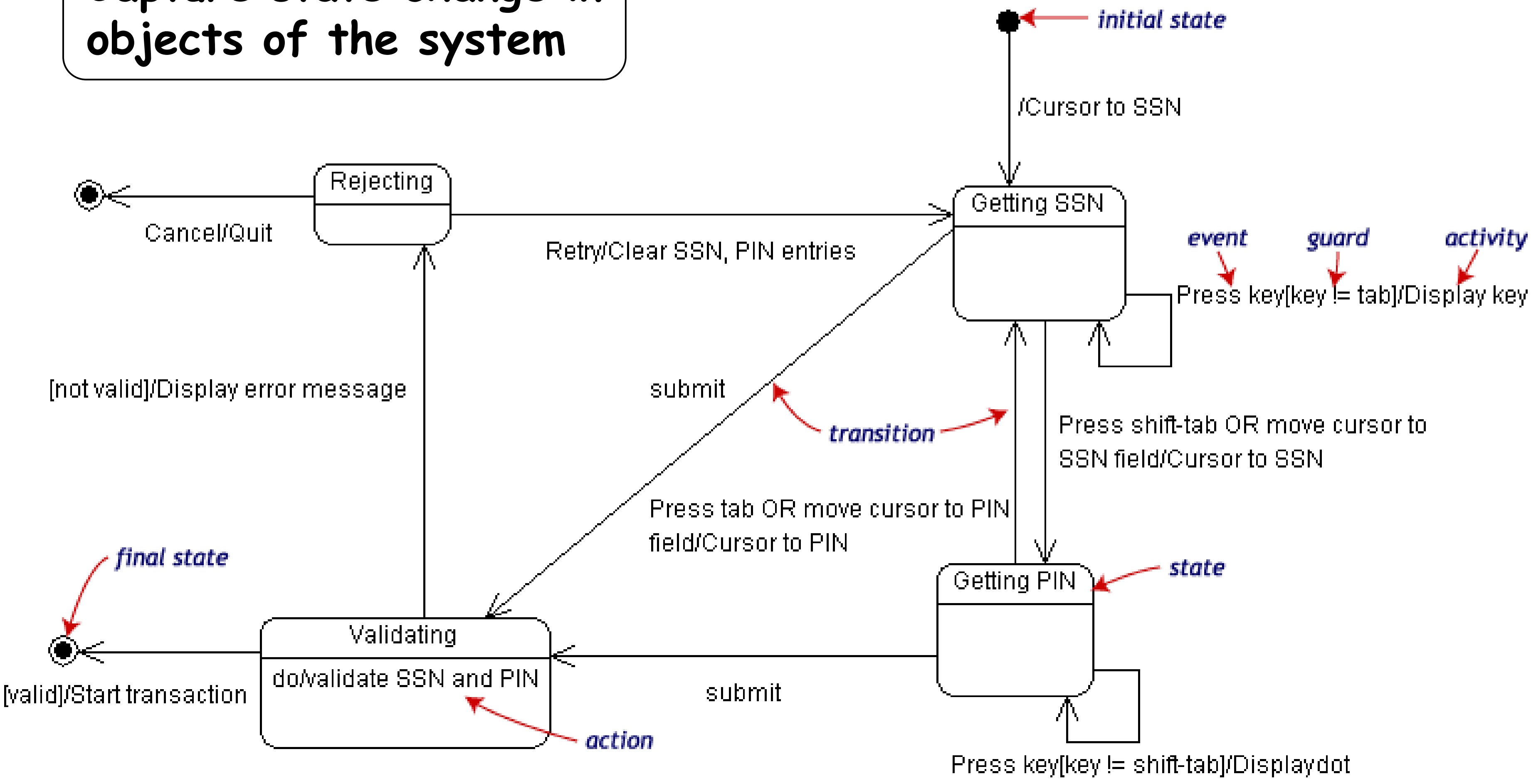


Capture how objects interact to achieve a goal

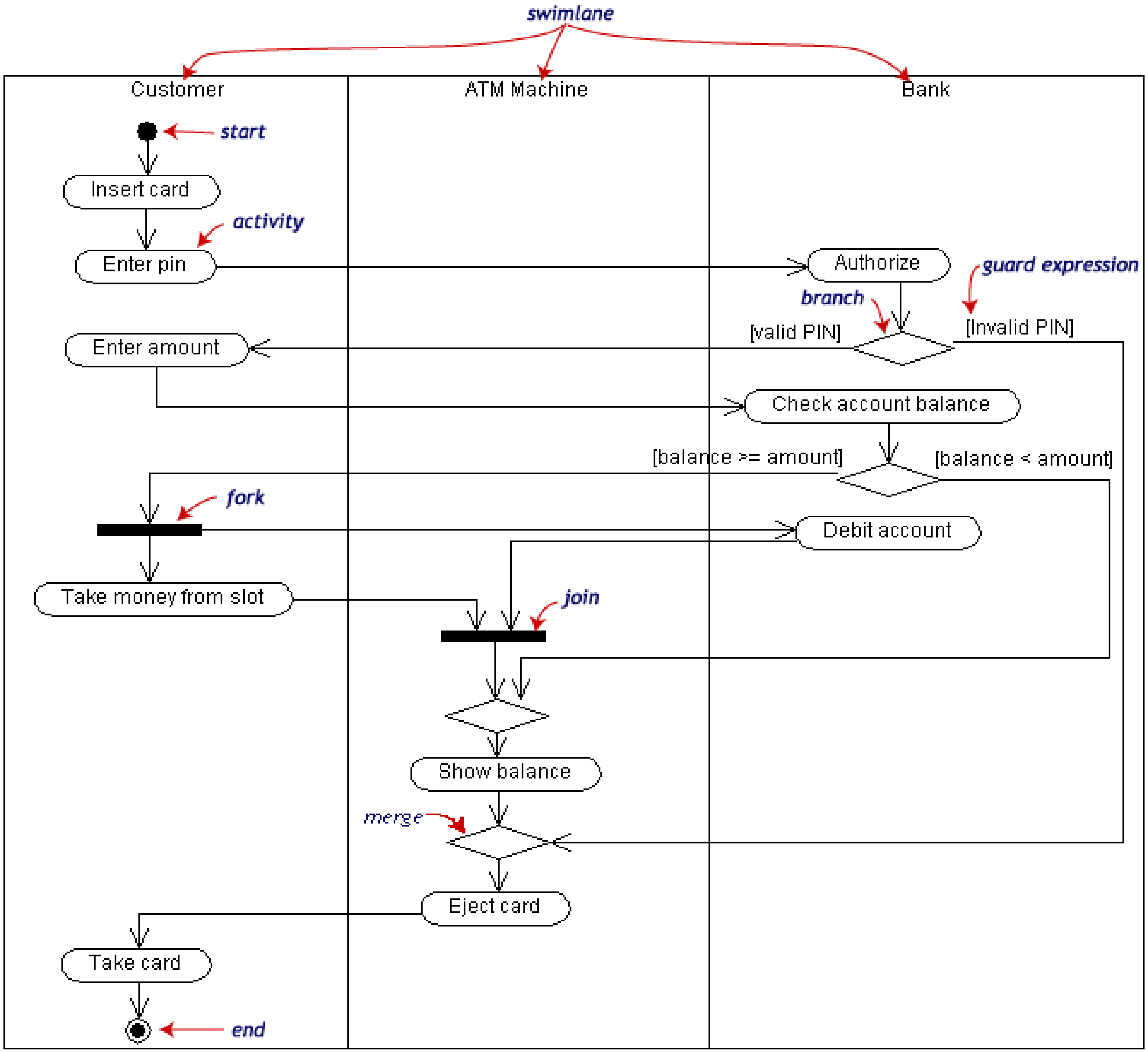


# UML: Statechart Diagrams

Capture state change in objects of the system

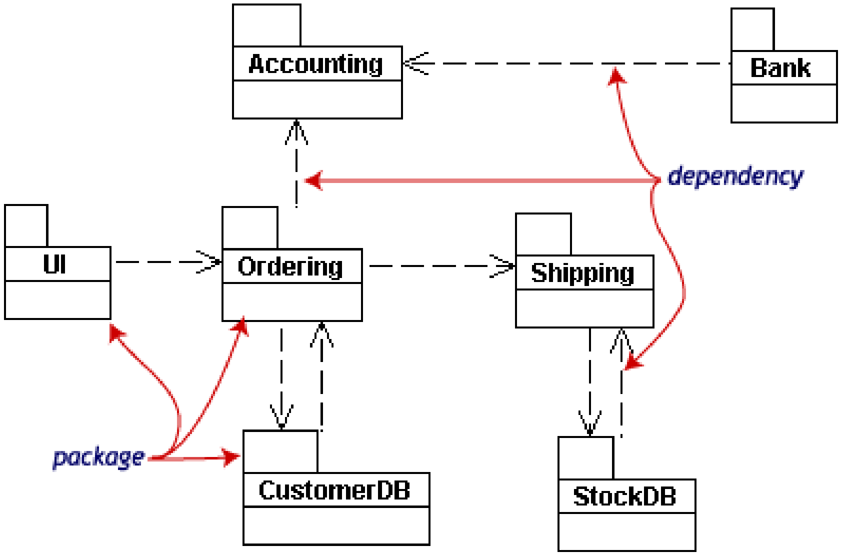

initial state
/Cursor to SSN
Rejecting
Cancel/Quit
Retry/Clear SSN, PIN entries
Getting SSN
event
guard
activity
Press key[key != tab]/Display key
[not valid]/Display error message
submit
transition
Press shift-tab OR move cursor to SSN field/Cursor to SSN
Press tab OR move cursor to PIN field/Cursor to PIN
final state
Getting PIN
state
Validating
do/validate SSN and PIN
[valid]/Start transaction
submit
action
Press key[key != shift-tab]/Displaydot

# UML: Activity Diagrams

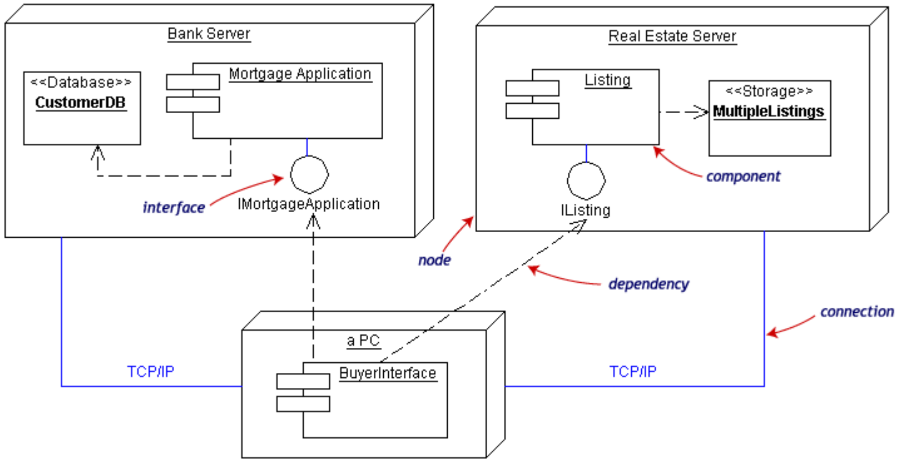


Capture the workflow in a situation

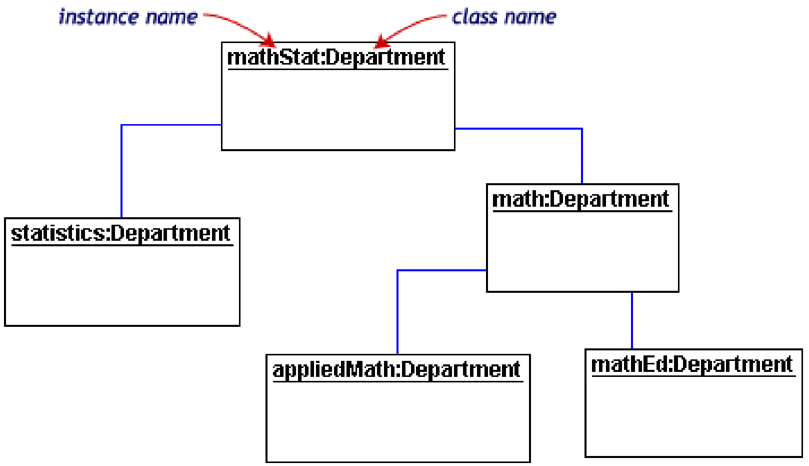
# UML: Other Diagrams

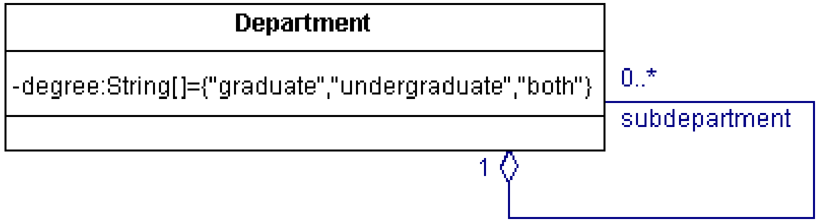


Package



Component and Deployment





Object

# References

- **Software Engineering**
  - Ian Sommerville. Software Engineering. 8th Edition, Addison-Wesley, 2007.
  - SWEBOK - Guide to the Software Engineering Body of Knowledge. 2004 Version, IEEE.
  - Bertrand Meyer. Software Engineering in the Academy. IEEE Computer, May 2001, pp. 28-35.
- **Dependability**
  - A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr. Basic Concepts and taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing 1(1):11-33, 2004.
  - Nancy G. Leveson. Safeware: System Safety and Computers. Addison-Wesley, 1995
  - Neil Story. Safety-Critical Computer Systems. Addison-Wesley, 1996.

# References

- **Other Case Studies**
  - The Ariane 5 Launcher Failure
  - The London Ambulance fiasco
  - Airbus Flight Control System
  - Medical Devices: The Therac-25
  - Lessons from 342 Medical Device Failures
  - A Collection of Software Bugs by Prof. Thomas Hackle
- **UML**
  - UML, Schaum's Outline Series, Simon Bennett, John Skelton and Ken Lunn, McGraw-Hill, 2001, ISBN 0-07-709673-8.
  - Online Tutorials
    - Practical UML: A Hands-On Introduction for Developers
    - Introduction to OMG's Unified Modeling Language (UML)
  - R. Miles, K. Hamilton. UML 2.0. O'Reilly 2006.
  - D. Pilone, N. Pitman. UML 2.0 in a nutshell. O'Reilly 2005.
  - James Rumbaugh, Ivar Jacobson and Grady Booch. The Unified Modeling Language Reference Manual. Second Edition, Addison-Wesley, 2004.
  - Perdita Stevens and Rob Pooley. Using UML: Software Engineering with Objects and Components. Addison-Wesley, 2000.