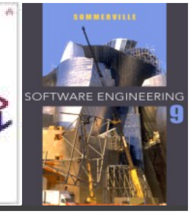
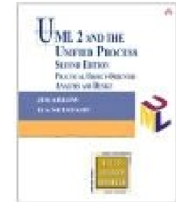


Non-functional Requirements Engineering

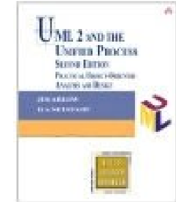
Lecture 3/Part 1

Outline




- ✧ Non-functional requirements classification
- ✧ Discussion of selected 15 non-functional requirements
- ✧ Non-functional requirements implementation
- ✧ UML Activity diagram

Functional and non-functional requirements



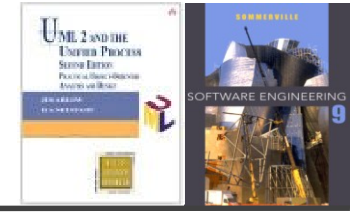
✧ Functional requirements

- Statements of **services the system should provide**, how the system should react to particular inputs and how the system should **behave** in particular situations.
- May state what the system should not do.

✧ Non-functional requirements

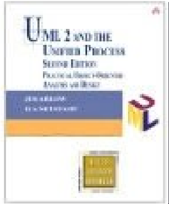
- Properties and **constraints on the services** offered by the system such as timing, reliability and security constraints, constraints on the development process, platform, standards, etc.
- **Product**, **organisational** and **external** requirements.
- Often apply to the **system as a whole** rather than individual features or services.

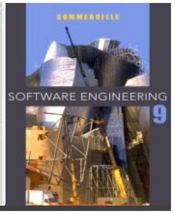
Non-functional requirements classification



- ✧ Product requirements
  - Requirements which specify that the delivered product must behave with a certain **quality** e.g. execution speed, reliability, etc.
- ✧ Organisational requirements
  - Requirements which are a consequence of **organisational policies and procedures** e.g. process standards used, implementation requirements, etc.
- ✧ External requirements
  - Requirements which arise from **factors which are external** to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Examples of non-functional requirements in the MHC-PMS



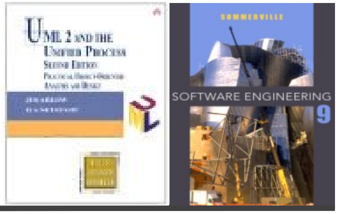


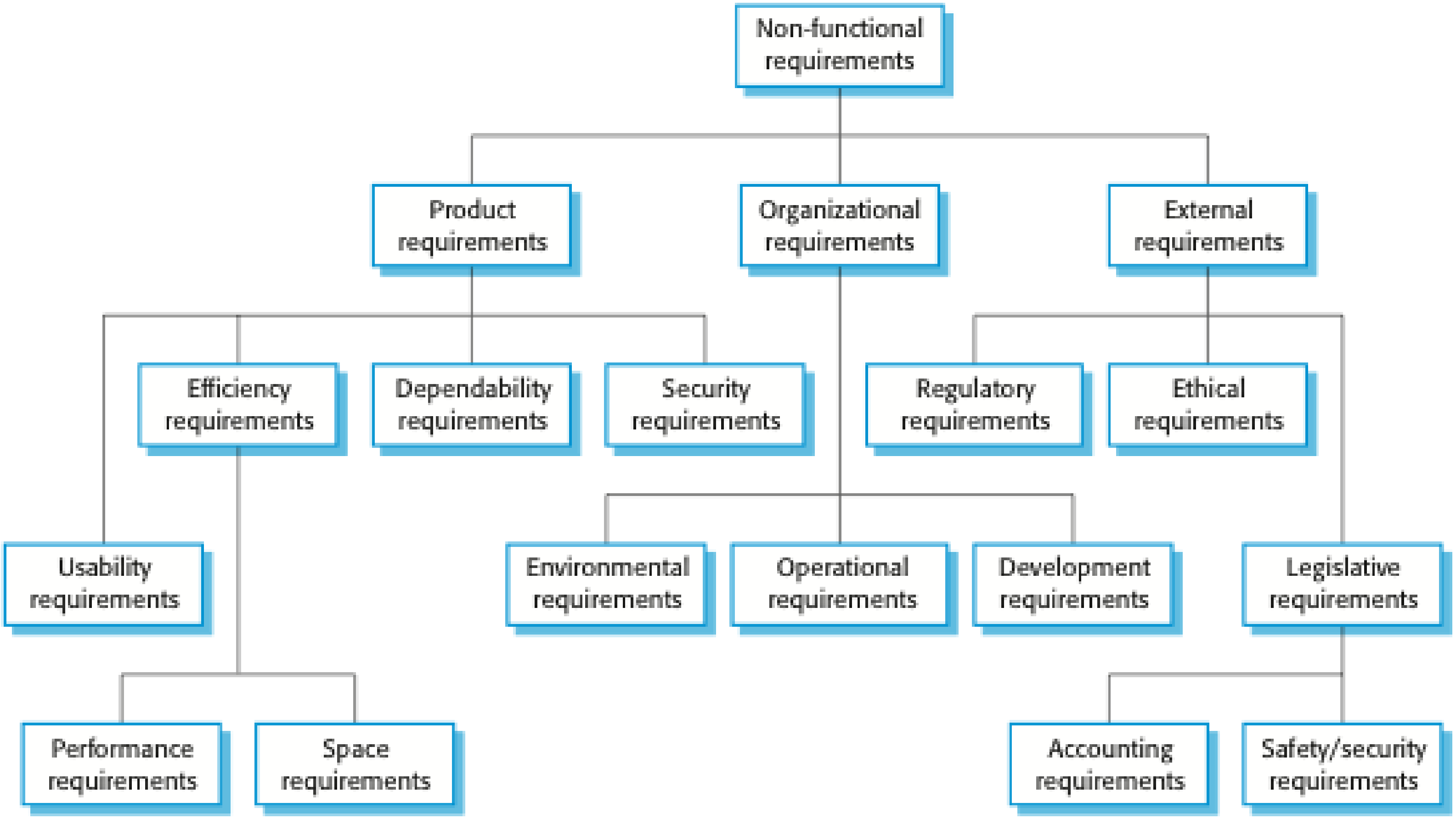
Product requirement
The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 08.30–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement
Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

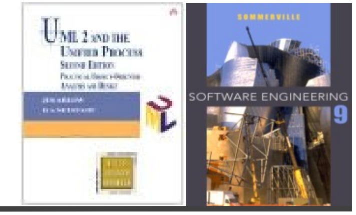
External requirement
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Types of non-functional requirements (excerpt)



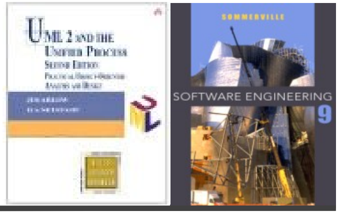


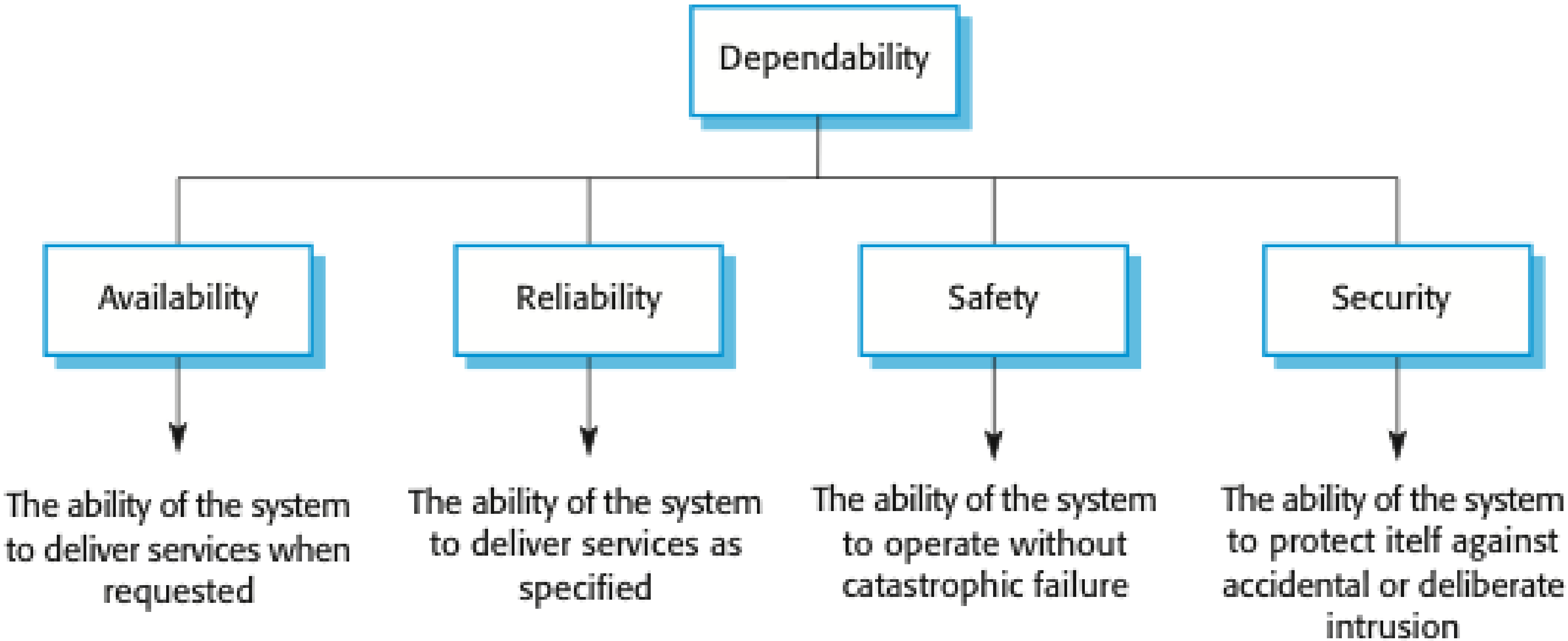
Product requirements



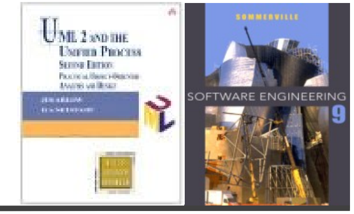
- ✧ Dependability
  - Availability
  - Reliability
  - Safety
  - Security
- ✧ Efficiency
  - Performance
  - Space/resource utilization
- ✧ Modifiability
- ✧ Testability
- ✧ Usability
- ✧ Resilience
- ✧ Robustness
- ✧ Understandability
- ✧ Adaptability
- ✧ Modularity
- ✧ Complexity
- ✧ Portability
- ✧ Reusability
- ✧ Learnability

Principal dependability attributes





Availability



- The probability that a system, at a point in time, will be operational and able to deliver the requested services

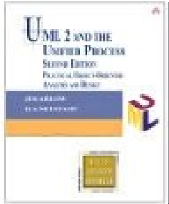
- Concerned with
  - How long the system should be operating without a failure.
  - How long a system is allowed to be out of operation.
- Can be expressed quantitatively
  - Using mean time to failure (MTTF) and repair (MTTR) as MTTF / (MTTF + MTTR).
  - I.e. availability of 0.999 means that the system is up and running for 99.9% of the time.

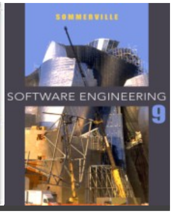
Reliability

- The probability of failure-free system operation over a specified time in a given environment for a given purpose

- Concerned with
  - How system fault/error/failure is detected.
  - How frequently system fault/error/failure may occur.
  - What happens when a fault/error/failure occurs.
- Can be expressed quantitatively
  - Using the probability of failure on demand (POFOD) within a single service or usage scenario execution, as 1 - POFOD.

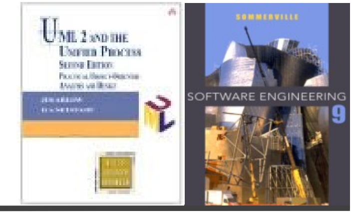
Reliability terminology





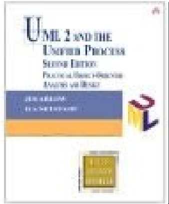
Term	Description
Human error or mistake	Human behavior that results in the introduction of faults into a system. For example, in the wilderness weather system, a programmer might decide that the way to compute the time for the next transmission is to add 1 hour to the current time. This works except when the transmission time is between 23.00 and midnight (midnight is 00.00 in the 24-hour clock).
System fault	A characteristic of a software system that can lead to a system error. The fault is the inclusion of the code to add 1 hour to the time of the last transmission, without a check if the time is greater than or equal to 23.00.
System error	An erroneous system state that can lead to system behavior that is unexpected by system users. The value of transmission time is set incorrectly (to 24.XX rather than 00.XX) when the faulty code is executed.
System failure	An event that occurs at some point in time when the system does not deliver a service as expected by its users. No weather data is transmitted because the time is invalid.

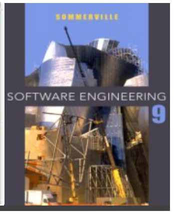
Availability and reliability



- It is sometimes possible to subsume system availability under system reliability
  - Obviously if a system is unavailable it is not delivering the specified system services.
- However, it is possible to have systems with low reliability that must be available.
  - So long as system failures can be repaired quickly and does not damage data, some system failures may not be a problem.
- Availability is therefore best considered as a separate attribute reflecting whether or not the system can deliver its services.

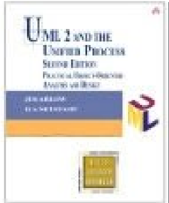
Safety

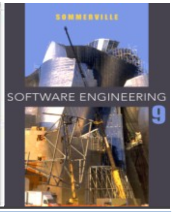




- Safety is a property of a system that reflects the system's ability to operate, normally or abnormally, without danger of causing human injury or death and without damage to the system's environment.
- It is important to consider software safety as most devices whose failure is critical now incorporate software-based control systems.
- Safety requirements are often exclusive requirements i.e. they exclude undesirable situations rather than specify required system services. These generate functional safety requirements.

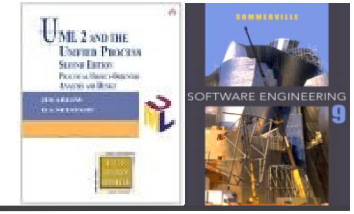
Safety terminology





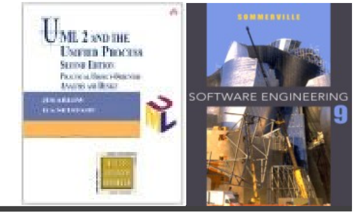
Term	Definition
Accident (or mishap)	An unplanned event or sequence of events which results in human death or injury, damage to property, or to the environment. An overdose of insulin is an example of an accident.
Hazard	A condition with the potential for causing or contributing to an accident. A failure of the sensor that measures blood glucose is an example of a hazard.
Damage	A measure of the loss resulting from a mishap. Damage can range from many people being killed as a result of an accident to minor injury or property damage. Damage resulting from an overdose of insulin could be serious injury or the death of the user of the insulin pump.
Hazard severity	An assessment of the worst possible damage that could result from a particular hazard. Hazard severity can range from catastrophic, where many people are killed, to minor, where only minor damage results. When an individual death is a possibility, a reasonable assessment of hazard severity is 'very high'.
Hazard probability	The probability of the events occurring which create a hazard. Probability values tend to be arbitrary but range from 'probable' (say 1/100 chance of a hazard occurring) to 'implausible' (no conceivable situations are likely in which the hazard could occur). The probability of a sensor failure in the insulin pump that results in an overdose is probably low.
Risk	This is a measure of the probability that the system will cause an accident. The risk is assessed by considering the hazard probability, the hazard severity, and the probability that the hazard will lead to an accident. The risk of an insulin overdose is probably medium to low.

Safety and reliability



- Safety and reliability are related but distinct
  - In general, reliability and availability are necessary but not sufficient conditions for system safety
- Reliability is concerned with conformance to a given specification and delivery of service
- Safety is concerned with ensuring system cannot cause damage irrespective of whether or not it conforms to its specification
- Unsafe reliable systems
  - If the system specification is incorrect then the system can behave as specified but still cause an accident.

Security

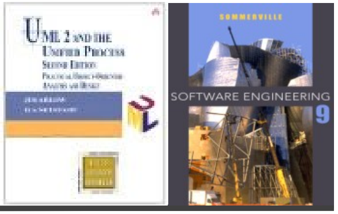


- A system property that reflects the system's ability to protect itself from accidental or deliberate external attack.

- Defends the system against:
  - Threats to the confidentiality of the system and its data
    - Can disclose information to people or programs that do not have authorization to access that information.
  - Threats to the integrity of the system and its data
    - Can damage or corrupt the software or its data.
  - Threats to the availability of the system and its data
    - Can restrict access to the system and data for authorized users.

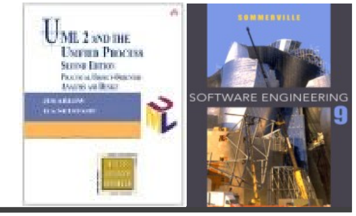
Security is an essential pre-requisite for availability, reliability and safety.

Security terminology



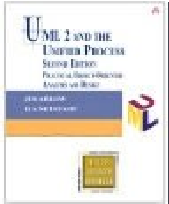
Term	Definition
Asset	Something of value which has to be protected (e.g. the patients records in MHC-PMS). The asset may be the software system itself or data used by that system.
Exposure	Possible loss or harm to a computing system, incl. e.g. the financial loss from patients' legal action or loss of reputation. This can be loss or damage to data, or can be a loss of time and effort if recovery is necessary after a security breach.
Vulnerability	A weakness in a computer-based system that may be exploited to cause loss or harm (e.g. weak password).
Attack	An exploitation of a system's vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage.
Threats	Circumstances that have potential to cause loss or harm. You can think of these as a system vulnerability that is subjected to an attack (e.g. guessing the weak password).
Control	A protective measure that reduces a system's vulnerability. Encryption is an example of a control that reduces a vulnerability of a weak access control system, or a password checking system in our example.

Dependability attribute dependencies



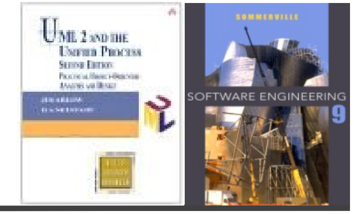
- ✧ Safe system operation depends on the system being available and operating reliably.
- ✧ A system may be unreliable because its data has been corrupted by an external attack.
- ✧ Denial of service attacks on a system are intended to make it unavailable.
- ✧ If a system is infected with a virus, you cannot be confident in its reliability or safety.

Performance



- ✧ Performance is about timing – response time to events (interrupts, messages, requests from users, or the passage of time).
  - For the Web-based financial system, the response might be the number of transactions that can be processed in a minute.
  - For the engine control system, the response might be the variation in the firing time.
- ✧ Highly sensitive to concurrency effects (number of users, shared resources), hardware, operating system implementation (e.g. scheduler strategy), etc.
- ✧ Often accompanied by characterization of throughput and resource utilization.

Modifiability



✧ Modifiability is about the cost of change.

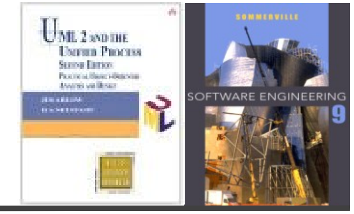
✧ What can change (the artifact)?

- The functions that the system computes, the platform the system exists on (the hardware, operating system, middleware, etc.), the environment within which the system operates, etc.

✧ When is the change made and who makes it (the environment)?

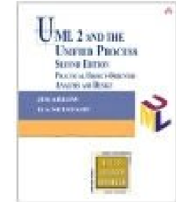
- During implementation (by modifying the source code), compile (using compile-time switches), build (by choice of libraries), configuration setup (by a range of techniques, including parameter setting) or execution (by parameter setting).
- By a developer, an end user, or a system administrator.

Testability



- Software testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.
  - At least 40% of the cost of developing well-engineered systems is taken up by testing. If the software architect can reduce this cost, the payoff is large.
- The response measures for testability deal with how effective the tests are in discovering faults and how long it takes to perform the tests to some desired level of coverage.
  - For a system to be properly testable, it must be possible to control each component's internal state and inputs and then to observe its outputs.

Usability




✧ Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides.

✧ It can be broken down into the following areas:

- Learning system features.
- Using a system efficiently.
- Minimizing the impact of errors.
- Adapting the system to user needs.
- Increasing confidence and satisfaction.

Organisational requirements

✧ Development requirements

- Programming language, development environment, process standards, time to market, rollout schedule, costs, etc.

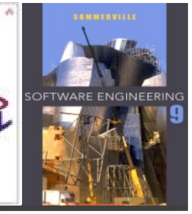
✧ Operational requirements

- Execution platform and other restrictions, system usage, projected lifetime, etc.

✧ Environmental requirements

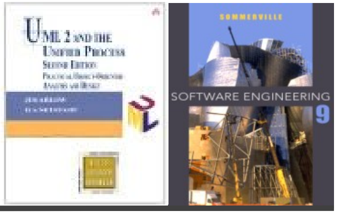
- Integration with legacy systems, targeted market, etc.

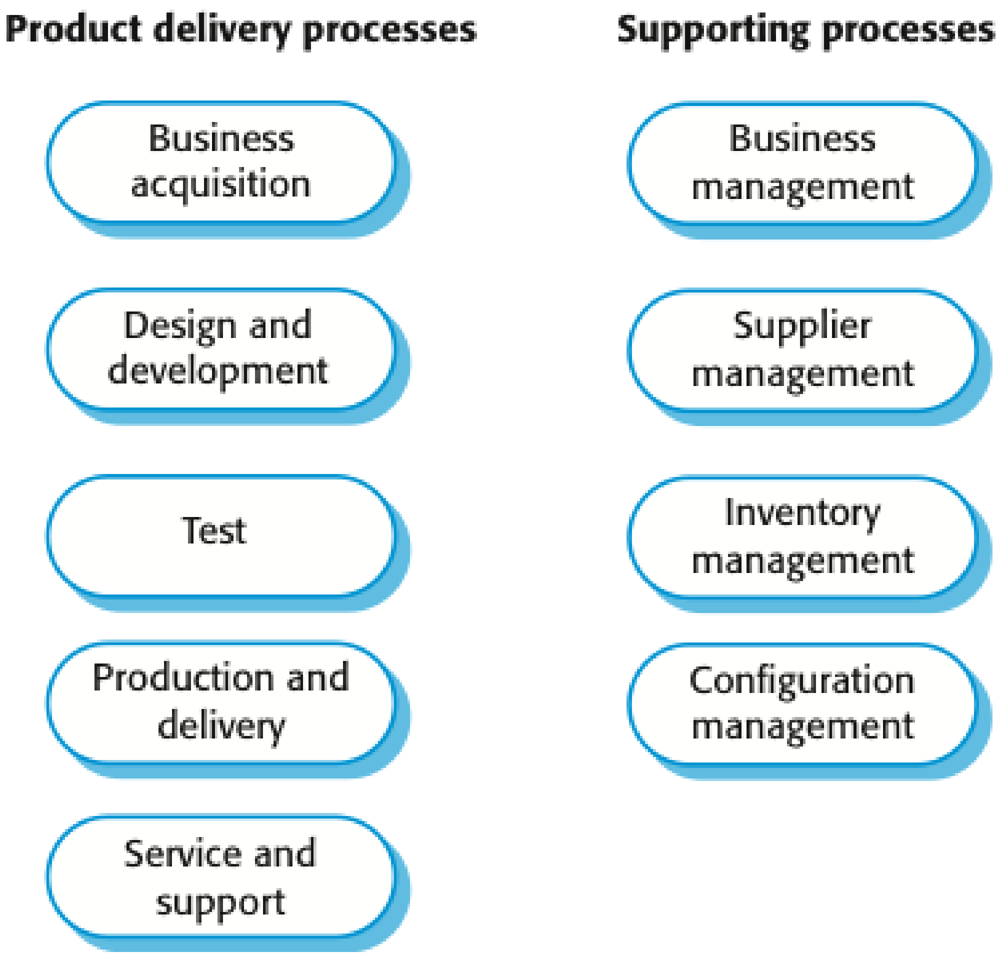
Process standards [development]



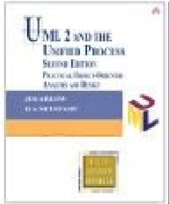
- ✧ Encapsulation of best practice
  - Avoids repetition of past mistakes.
- ✧ Provide continuity
  - New staff can understand the organisation by understanding the standards that are used.
- ✧ ISO 9001
  - International set of standards that can be used as a basis for developing quality management systems.
  - Applies to organizations that design, develop and maintain products, including software.
  - Sets out general quality principles, describes quality processes and lays out the organizational procedures that should be defined.

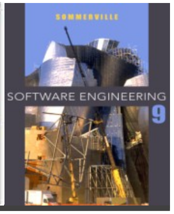
ISO 9001 core processes





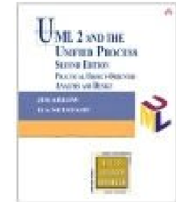
Time to market [development]





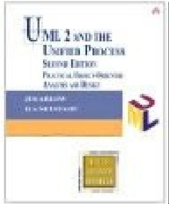
- ✧ If there is competitive pressure or a short window of opportunity for a system or product, development time becomes important.
- ✧ This in turn leads to pressure to buy or otherwise re-use existing elements.
- ✧ Time to market is often reduced by using prebuilt elements such as commercial off-the-shelf (COTS) products or elements re-used from previous projects.
- ✧ The ability to insert or deploy a subset of the system depends on the decomposition of the system into elements.

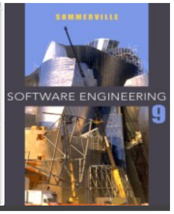
Rollout schedule [development]



- ✧ If a product is to be introduced as base functionality with many features released later, the flexibility and customizability of the architecture are important.
- ✧ Particularly, the system must be constructed with ease of expansion and contraction in mind.

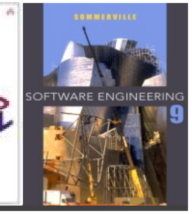
Cost and benefit [development]





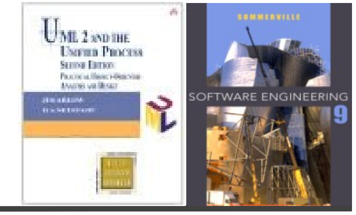
- ✧ The development effort will naturally have a budget that must not be exceeded.
- ✧ Different designs will yield different development costs.
  - For instance, an implementation that relies on technology (or expertise with a technology) not resident in the developing organization will be more expensive to realize than one that takes advantage of assets already inhouse.
  - An implementation that is highly flexible will typically be more costly to build than one that is rigid (although it will be less costly to maintain and modify).

Projected lifetime of the system [operational]



- If the system is intended to have a long lifetime, modifiability, scalability, and portability become important.
- But building in the additional infrastructure (such as a layer to support portability) will usually compromise time to market.
- On the other hand, a modifiable, extensible product is more likely to survive longer in the marketplace, extending its lifetime.

Integration with legacy systems [environmental]

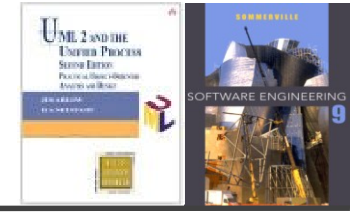


- ✧ If the new system has to integrate with existing systems, care must be taken to define appropriate integration mechanisms.
- ✧ This property is clearly of marketing importance but has substantial design implications.
  - For example, the ability to integrate a legacy system with an HTTP server to make it accessible from the Web has been a marketing goal in many corporations over the past decade.
  - The architectural constraints implied by this integration must be analyzed.

Targeted market [environmental]

- For general-purpose (mass-market) software, the platforms on which a system runs as well as its feature set will determine the size of the potential market.
- Thus, portability and functionality are key to market share.
- To attack a large market with a collection of related products, a **product line approach** should be considered in which a core of the system is common (frequently including provisions for portability) and around which layers of software of increasing specificity are constructed.

External requirements



- ✧ Regulatory requirements
- ✧ Ethic requirements
- ✧ Legislative requirements
  - Accounting legislative
  - Safety/Security legislative

Non-functional requirements implementation

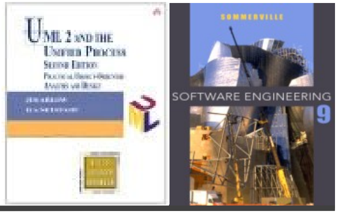
✧ Non-functional requirements may affect the overall architecture of a system rather than the individual components.

- For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.

✧ A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.

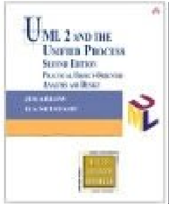
- It may also generate requirements that restrict existing requirements.

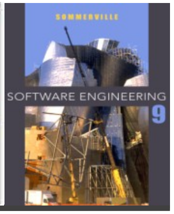
Design for non-functional requirements



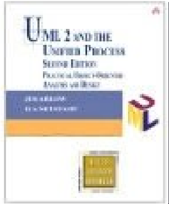
- Most commonly related to product requirements (non-functional product qualities)
- Specific to each non-functional product attribute
  - Tactics discussed later in the course.
- Design-time reliability/performance/… prediction
  - Support of early design decisions based on the prediction of non-functional product qualities early in system design.
  - Commonly based on annotated UML Activity diagrams.

Key points





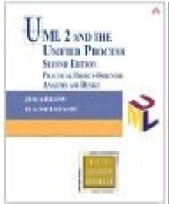
- We have discussed examples of product, operational and external non-functional requirements.
- Specific attention has been paid to:
  - Availability
  - Reliability
  - Safety
  - Security
  - Performance
  - Modifiability
  - Testability
  - Usability

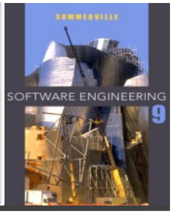


UML Activity Diagram

Lecture 3/Part 2

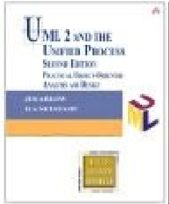
What are activity diagrams?





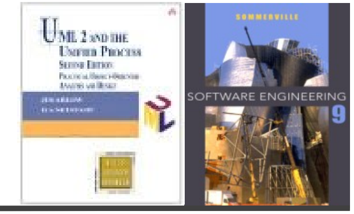
- Activity diagrams are "OO flowcharts"
- They allow us to model a process as a collection of nodes and edges between those nodes
- Use activity diagrams to model the behavior of:
  - use cases
  - classes
  - interfaces
  - components
  - collaborations
  - operations and methods
  - business processes

Activities

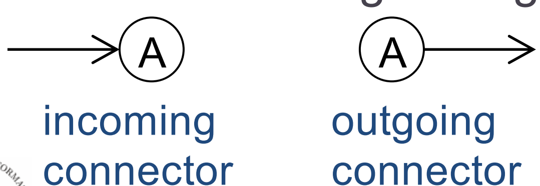


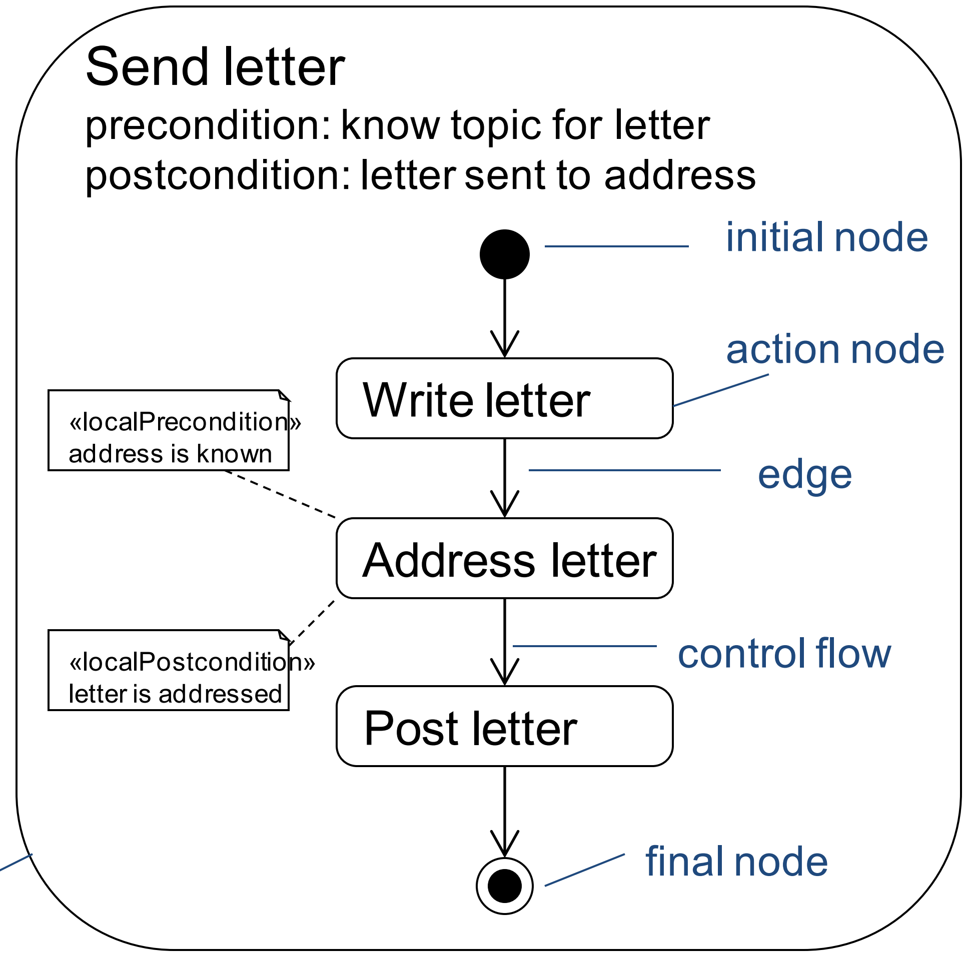
- Activities are networks of nodes connected by edges
- There are three categories of node:
  - **Action nodes** - represent discrete units of work that are atomic within the activity
  - **Control nodes** - control the flow through the activity
  - **Object nodes** - represent the flow of objects around the activity
- Edges represent flow through the activity
- There are two categories of edge:
  - **Control flows** - represent the flow of control through the activity
  - **Object flows** - represent the flow of objects through the activity

Activity diagram syntax

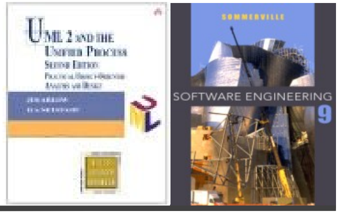


- Activities are networks of *nodes* connected by *edges*
  - The control flow is a type of edge
- Activities usually start in an *initial node* and terminate in a *final node*
- Activities can have preconditions and postconditions
- When an action node finishes, it emits a token that may traverse an edge to trigger the next action
  - This is sometimes known as a *transition*
- You can break an edge using:

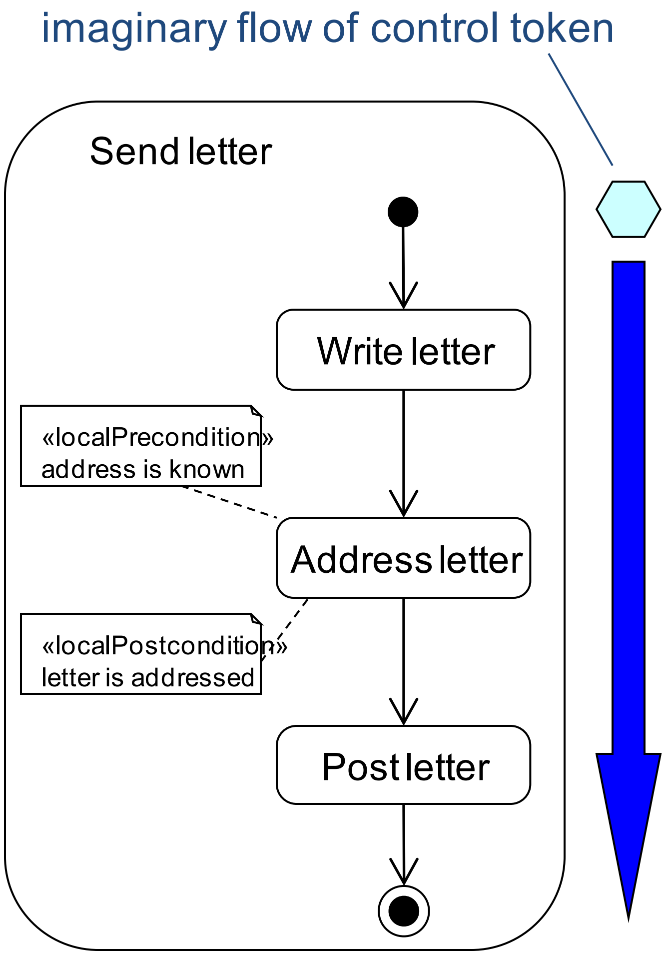




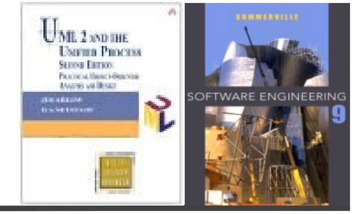
Activity diagram semantics



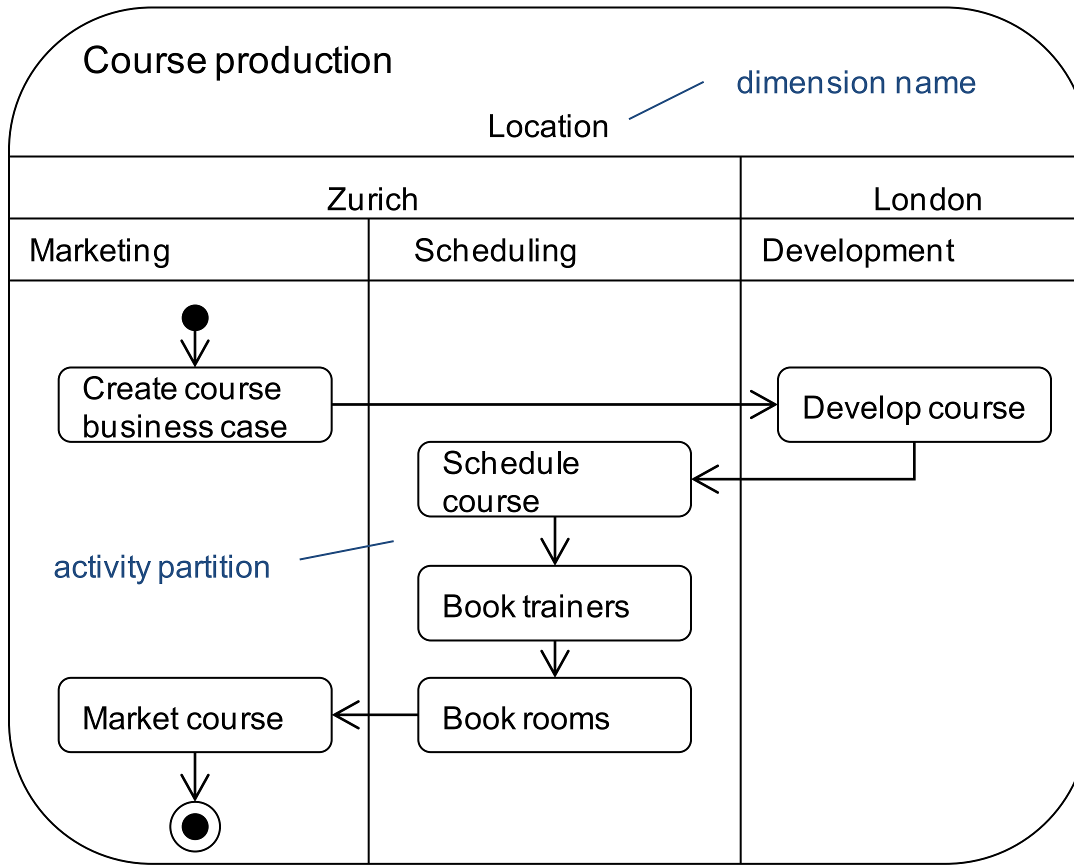
- The *token game*
  - Token – an object, some data or a focus of control
  - Imagine tokens flowing around the activity diagram
- Tokens traverse from a source node to a target node via an edge
  - The source node, edge and target node may all have constraints controlling the movement of tokens
  - All constraints *must* be satisfied before the token can make the traversal
- A node executes when:
  - It has tokens on **all of its input edges** AND these tokens satisfy predefined conditions (see later)
- When a node starts to execute it takes tokens off its input edges
- When a node has finished executing it offers tokens on its output edges



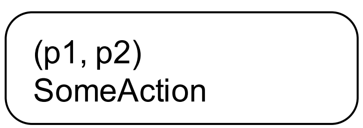
Activity partitions



- Each activity partition represents a high-level grouping of a set of related actions
  - Partitions can be hierarchical
  - Partitions can be vertical, horizontal or both
- Partitions can refer to many different things e.g. business organisations, classes, components and so on
- If partitions can't be shown clearly using parallel lines, put their name in brackets directly above the name of the activities

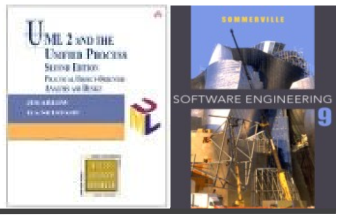


nested partitions

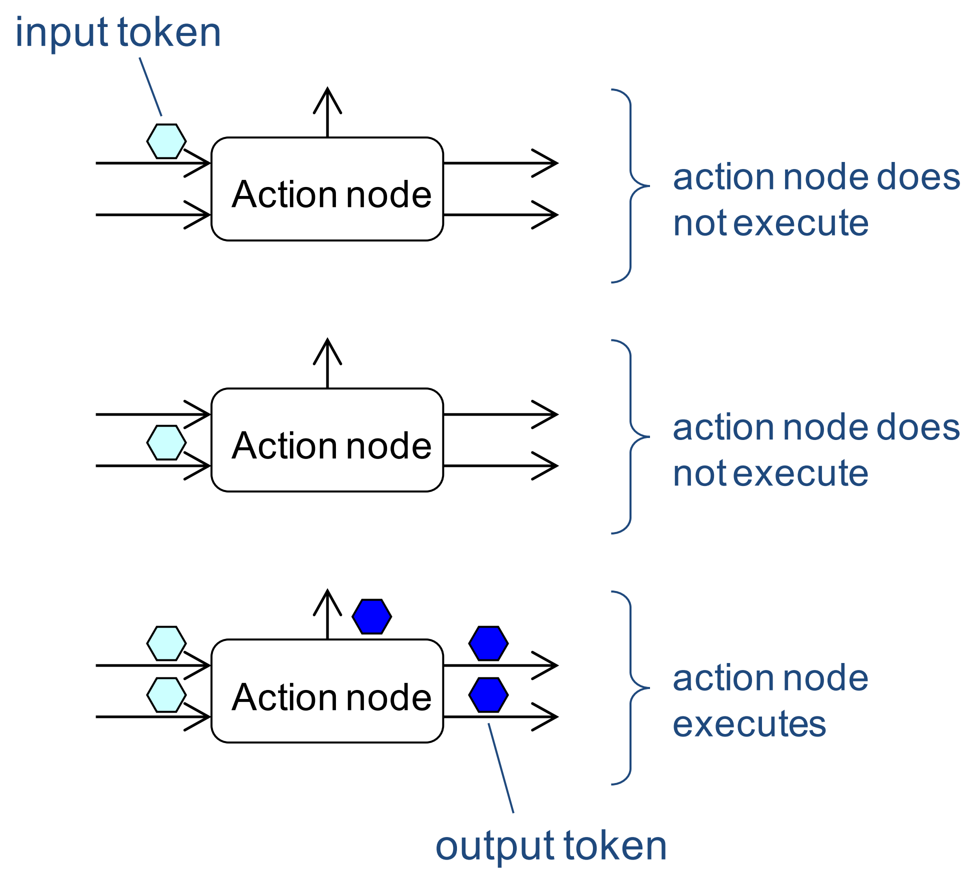


multiple partitions

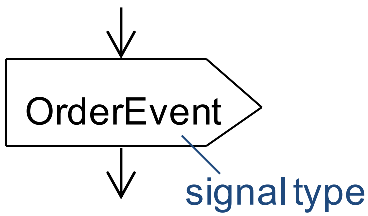
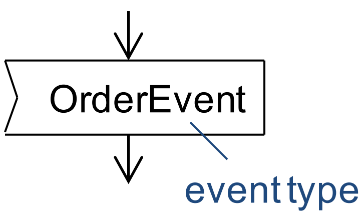
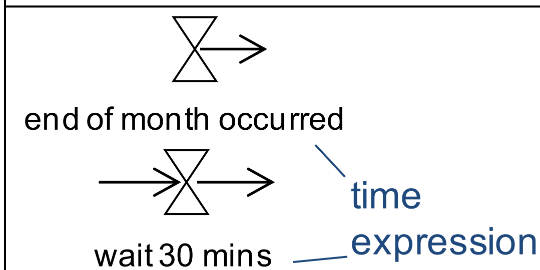
Action nodes



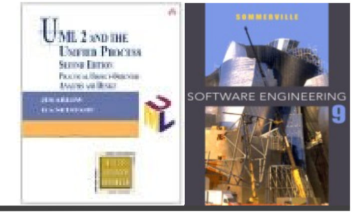
- Action nodes offer a token on *all* of their output edges when:
  - There is a token *simultaneously* on each input edge
  - The input tokens satisfy all preconditions specified by the node
- Action nodes:
  - Perform an implicit fork on their output edges when they have finished executing



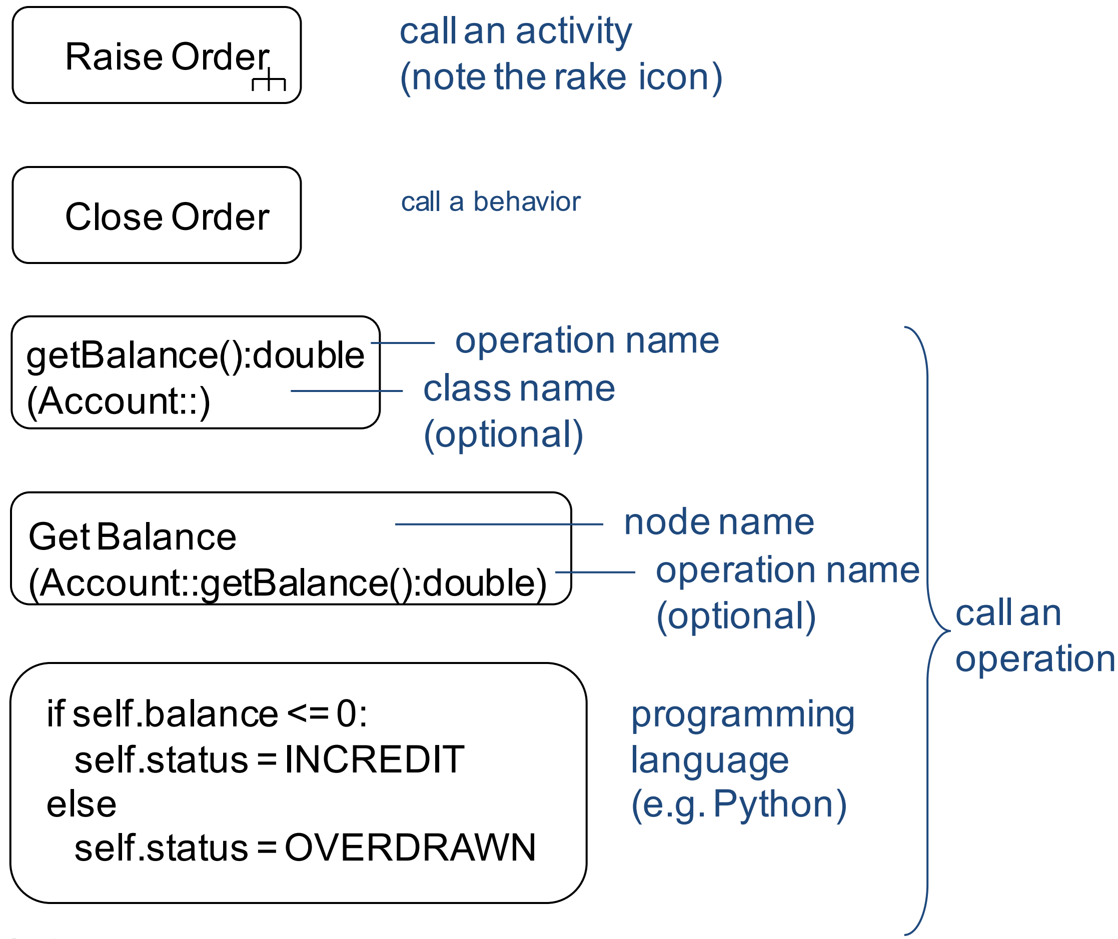
Types of action node

action node syntax	action node semantics
	Call action - invokes an activity, a behavior or an operation. The most common type of action node. See next slide for details.
	Send signal action - sends a signal asynchronously. The sender *does not* wait for confirmation of signal receipt. It may accept input parameters to create the signal
	Accept event action - waits for events detected by its owning object and offers the event on its output edge. Is enabled when it gets a token on its input edge. If there is *no* input edge it starts when its containing activity starts and is *always* enabled.
	Accept time event action - waits for a set amount of time. Generates time events according to it's time expression.

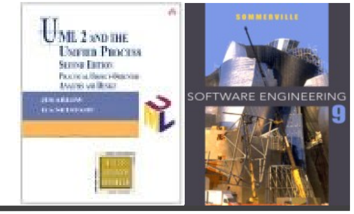
Call action node syntax



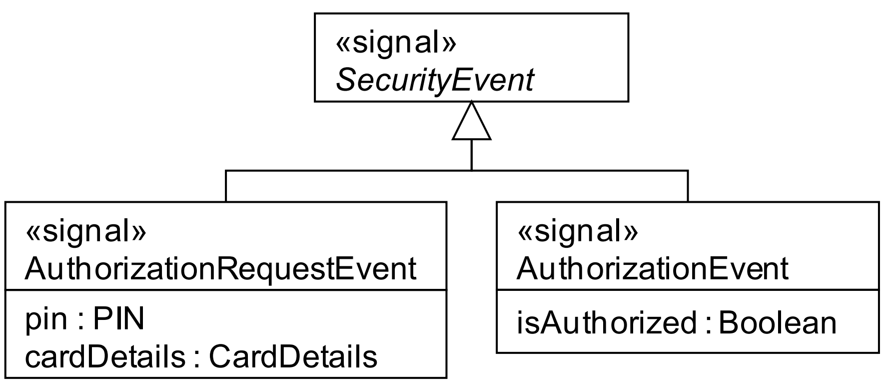
- The most common type of node
- Call action nodes may invoke:
  - an activity
  - a behavior
  - an operation
- They may contain code fragments in a specific programming language
  - The keyword 'self' refers to the context of the activity that owns the action

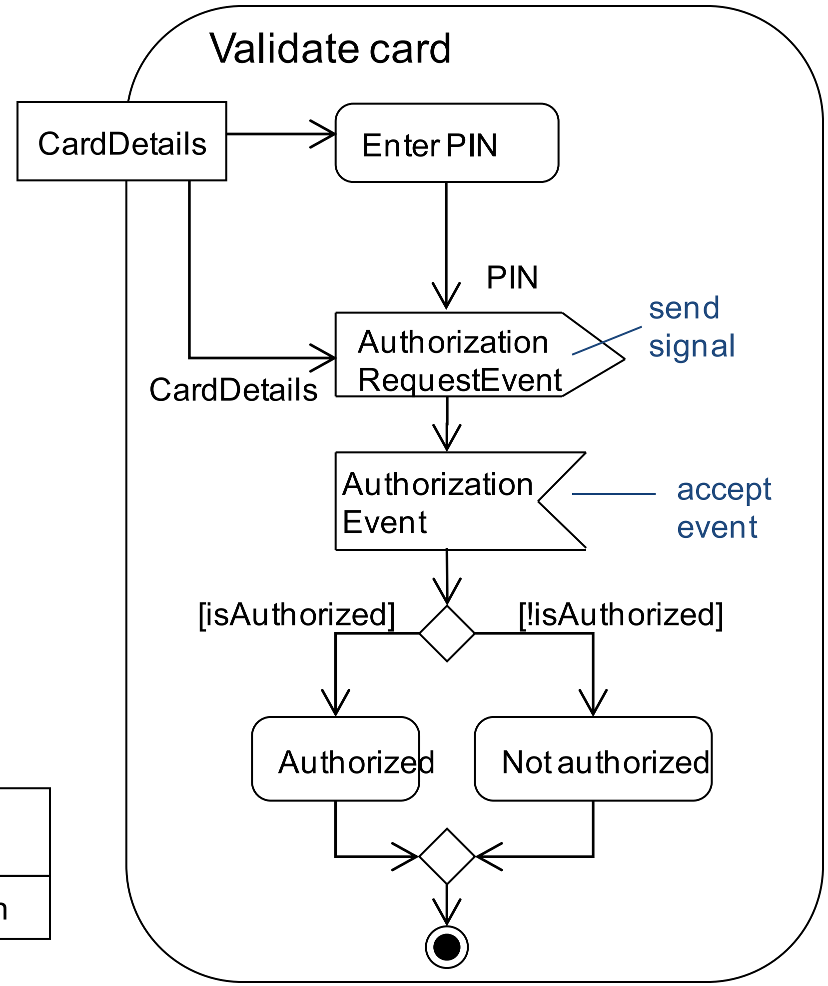


Sending signals and accepting events

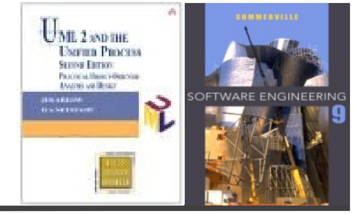


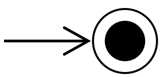
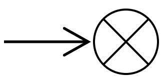
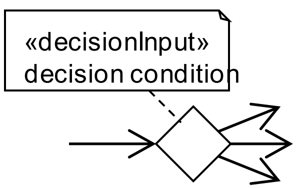
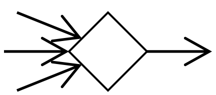
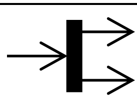
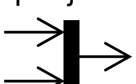
- Signals represent information passed asynchronously between objects
  - This information is modelled as attributes of a signal
  - A signal is a classifier stereotyped «signal»
- The accept event action asynchronously accepts event triggers which may be signals or other objects





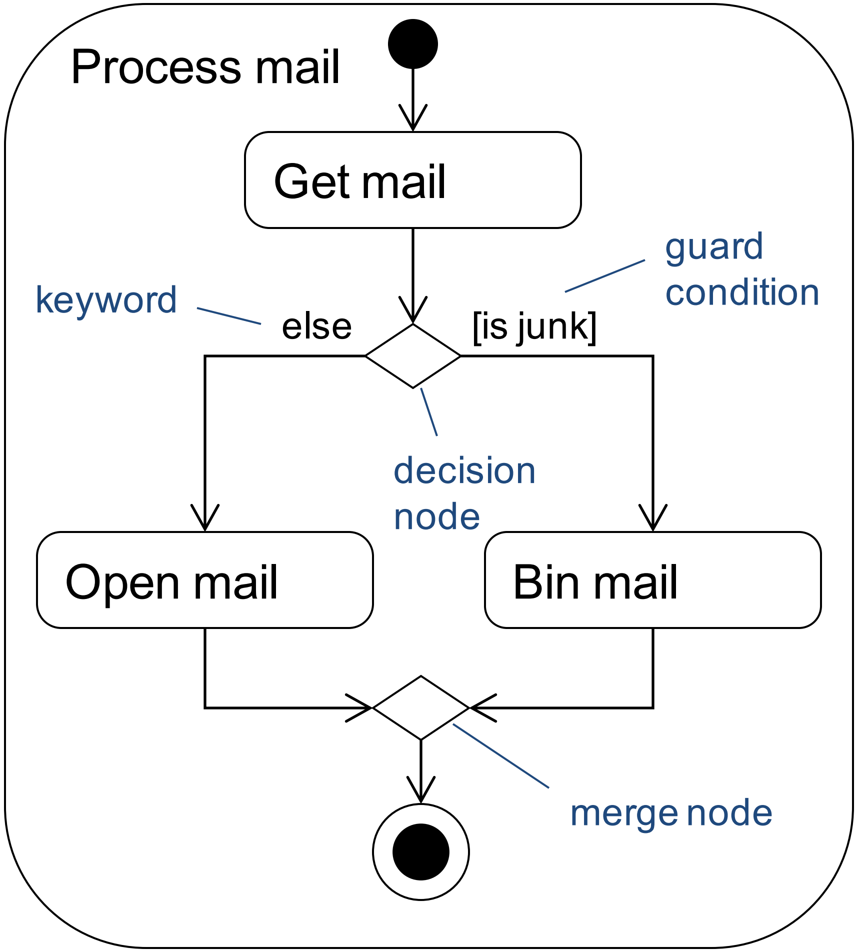
Control nodes



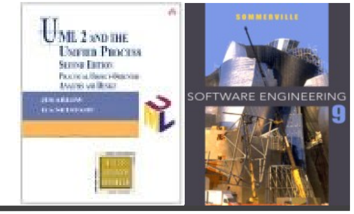
control node syntax	control node semantics	
	Initial node – indicates where the flow starts when an activity is invoked	
	Activity final node – terminates an activity	Final nodes
	Flow final node – terminates a specific flow within an activity. The other flows are unaffected	Final nodes
	Decision node– guard conditions on the output edges select one of them for traversal May optionally have inputs defined by a «decisionInput»	See examples on next two slides
	Merge node – selects *one* of its input edges	See examples on next two slides
	Fork node – splits the flow into multiple concurrent flows	See examples on next two slides
{join spec} 	Join node – synchronizes multiple concurrent flows May optionally have a join specification to modify its semantics	See examples on next two slides

Decision and merge nodes

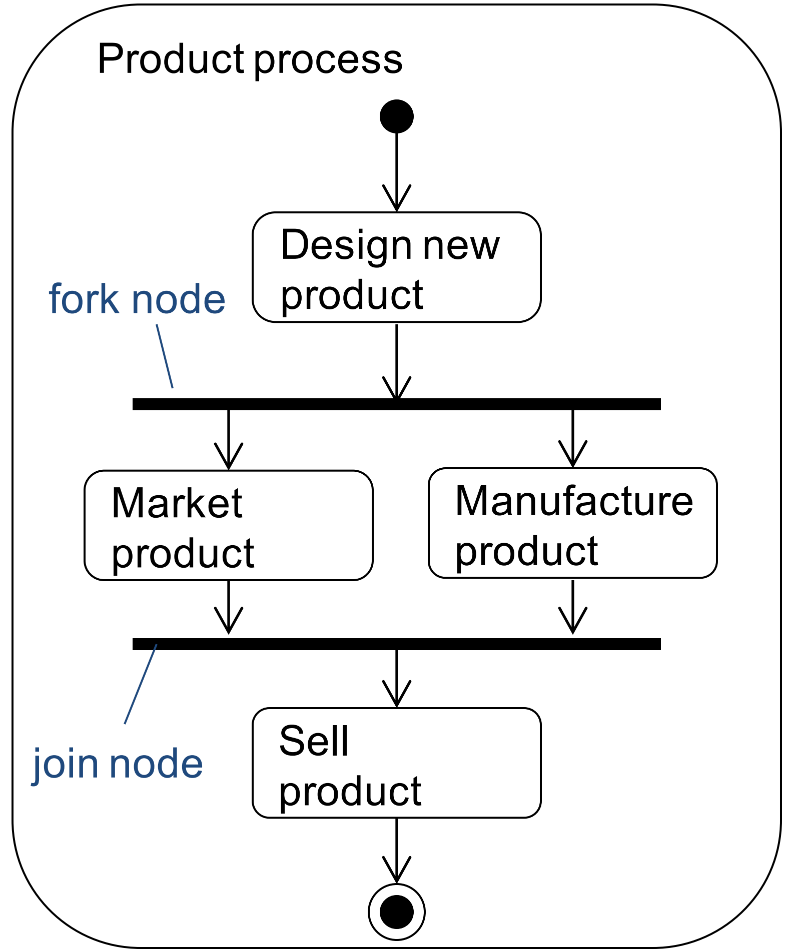
- A decision node is a control node that has one input edge and two or more alternate output edges
  - Each edge out of the decision is protected by a *guard condition*
  - guard conditions must be mutually exclusive
  - The edge can be taken if and only if the guard condition evaluates to true
  - The keyword *else* specifies the path that is taken if *none* of the guard conditions are true
- A merge node accepts one of several alternate flows
  - It has two or more input edges and exactly one output edge



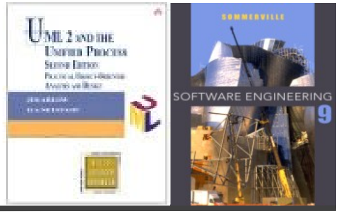
Fork and join nodes – concurrency



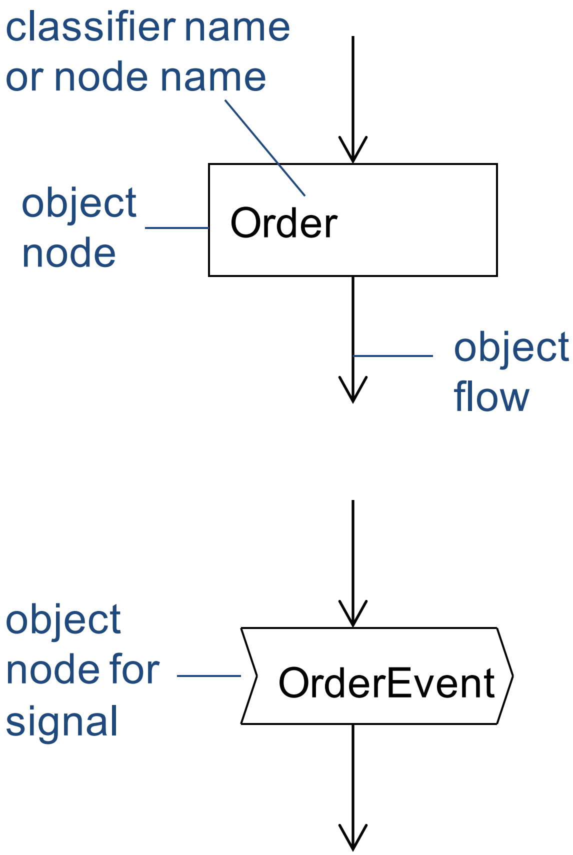
- Forks nodes model concurrent flows of work
  - Tokens on the single input edge are replicated at the multiple output edges
- Join nodes synchronize two or more concurrent flows
  - Joins have two or more incoming edges and exactly one outgoing edge
  - A token is offered on the outgoing edge when there are tokens on *all* the incoming edges i.e. when the concurrent flows of work have all finished



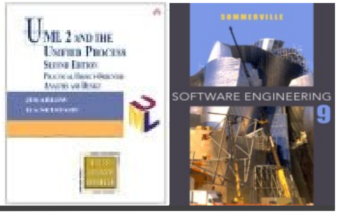
Object nodes



- Object nodes indicate that instances of a particular classifier may be available
  - If no classifier is specified, then the object node can hold any type of instance
- Multiple tokens can reside in an object node *at the same time*
  - The upper bound defines the maximum number of tokens (infinity is the default)
- Tokens are presented to the single output edge according to an ordering:
  - FIFO – first in, first out (the default)
  - LIFI – last in, first out
  - Modeler defined – a selection criterion is specified for the object node



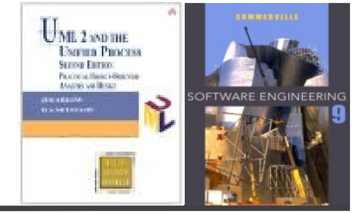
Object node syntax

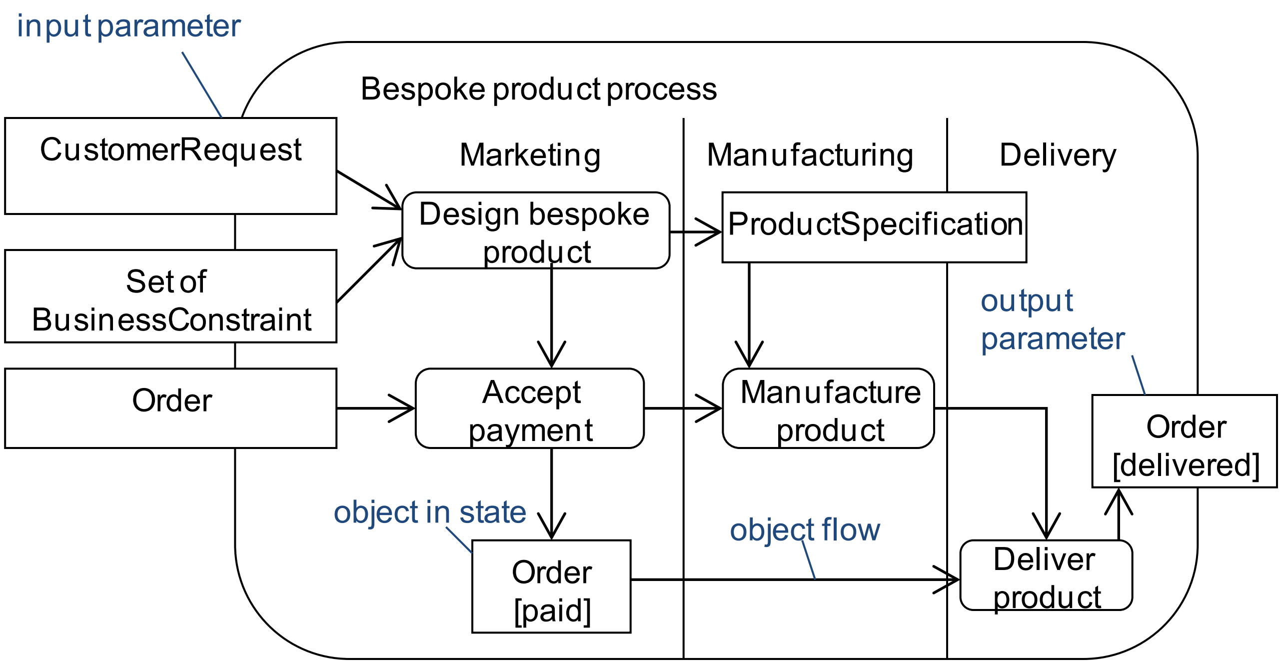


- Object nodes have a flexible syntax. You may show:
  - upper bounds
  - ordering
  - sets of objects
  - selection criteria
  - object in state

Order	order objects may be available
Order {upperBound = 12}	zero to 12 Order objects may be available
Order {ordering = LIFO}	last Order object in is the first out (FIFO is the default)
Set of Order	sets of Order objects may be available
«selection» monthRaised = "Dec" - - - Order	Order objects raised in December may be available
Order [open]	select Order objects in the open state

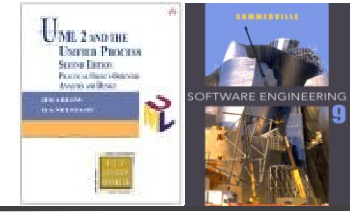
Activity parameters

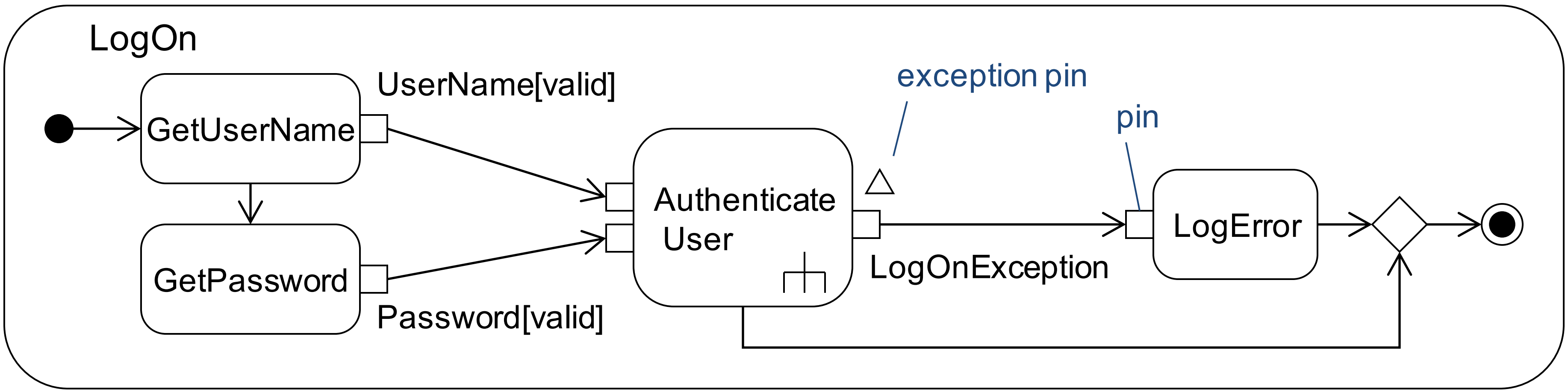




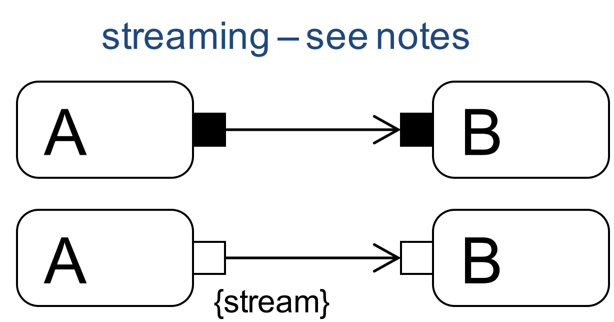
- Object nodes can provide input and output parameters to activities
  - Input parameters have one or more output object flows into the activity
  - Output parameters have one or more input object flows out of the activity
- Draw the object node overlapping the activity boundary

Pins



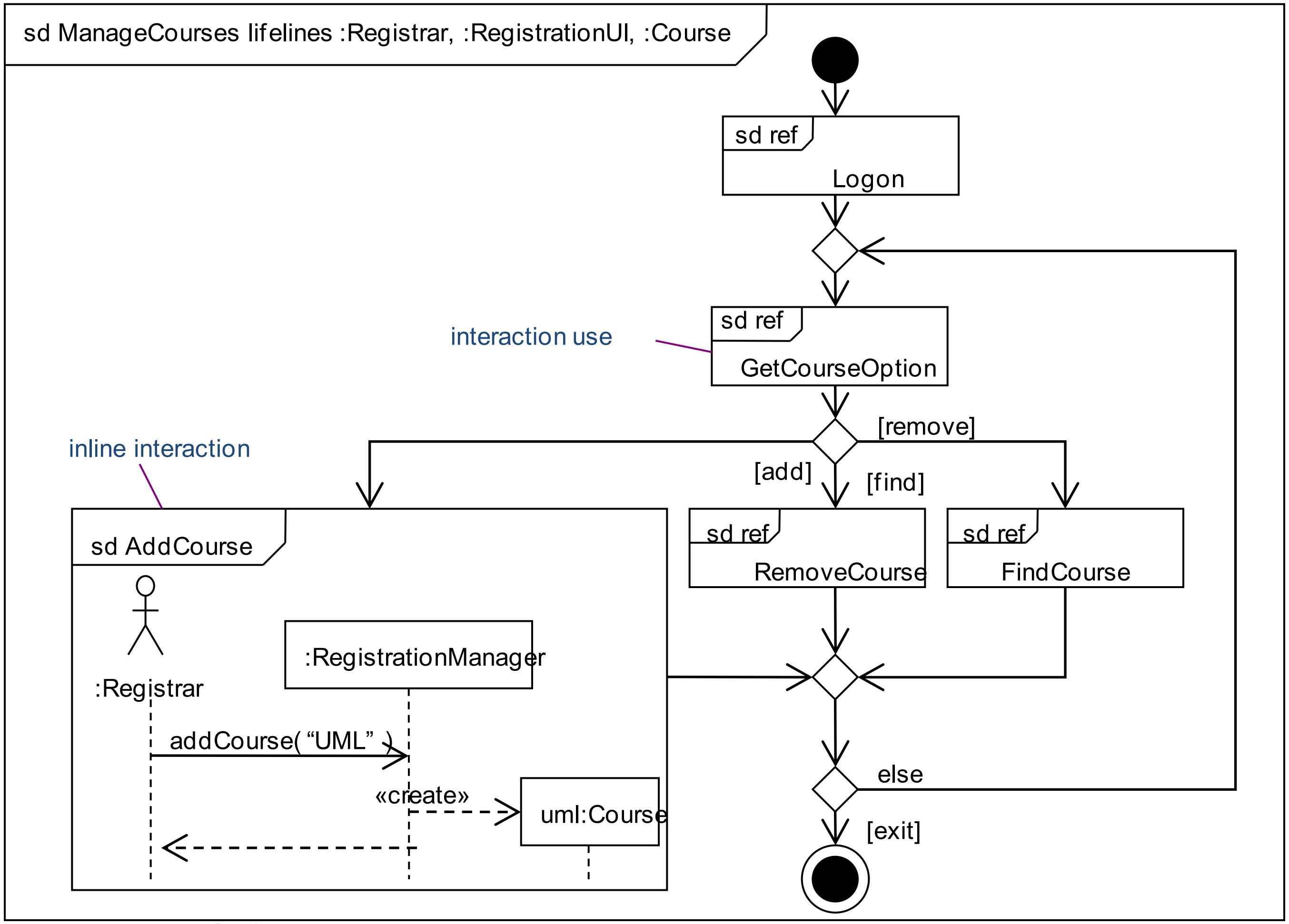


- Pins are object nodes for inputs to, and outputs from, actions
  - Same syntax as object nodes
  - Input pins have exactly one input edge
  - Output pins have exactly one output edge
  - Exception pins are marked with an equilateral triangle
  - Streaming pins are filled in black or marked with {stream}

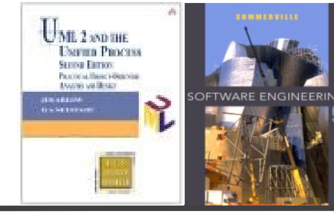


Interaction overview diagrams

- Model the high level flow of control between interactions
- Show interactions and interaction occurrences
- Have activity diagram syntax



Key points



- Activity diagrams can model flows of activities using:
  - Activities and connectors
  - Activity partitions
  - Action nodes
    - Call action node
    - Send signal/accept event action node
    - Accept time event action node
  - Control nodes
    - Decision and merge
    - Fork and join
  - Object nodes
    - Input and output parameters
    - Pins
- Interaction overview diagrams as their advanced feature