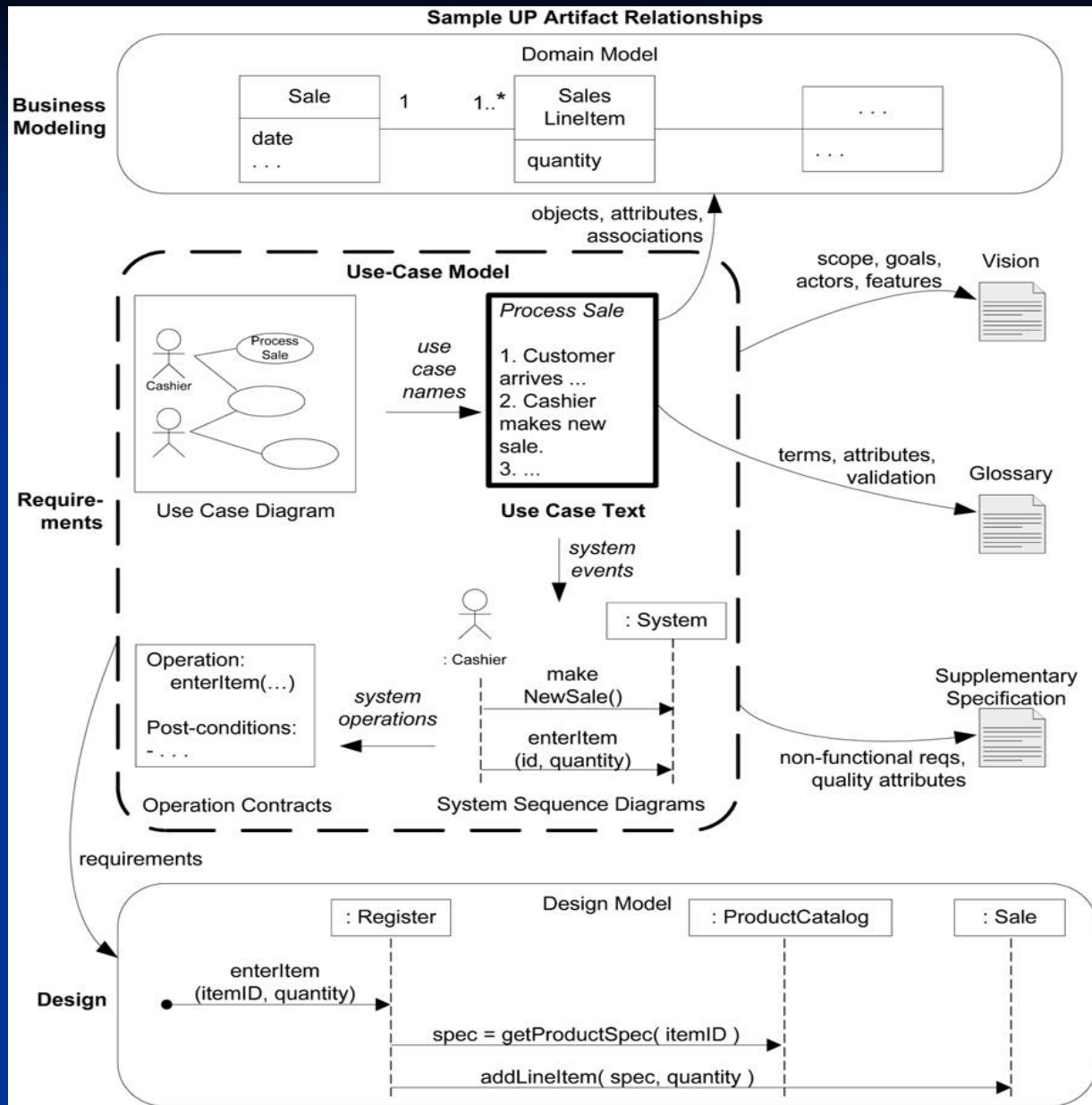


- **The Agenda for this lecture:**
 - **Unified Process:** Phases, Disciplines, and Artifacts
 - **Gathering Requirements**
 - **Use-Cases**
 - **System Sequence Diagrams**

Unified Process: Phases, Disciplines, and Artifacts

What are the different artifacts in
created in UP?



What is UP? Are Other Methods Complementary?

- What is the SDP?
 - A software development process describes an approach to building, deploying, and possibly maintaining software.
 - Documentation for who can do what, how and when
- The Unified Process has emerged as a popular iterative software development process for building object-oriented systems.
- The Rational Unified Process or RUP, a detailed refinement of the Unified Process, has been widely adopted.

What is UP? Are Other Methods Complementary?

- The UP is very flexible and open, and encourages including skillful practices from other iterative methods, such as from Extreme Programming (XP), Scrum, and so forth

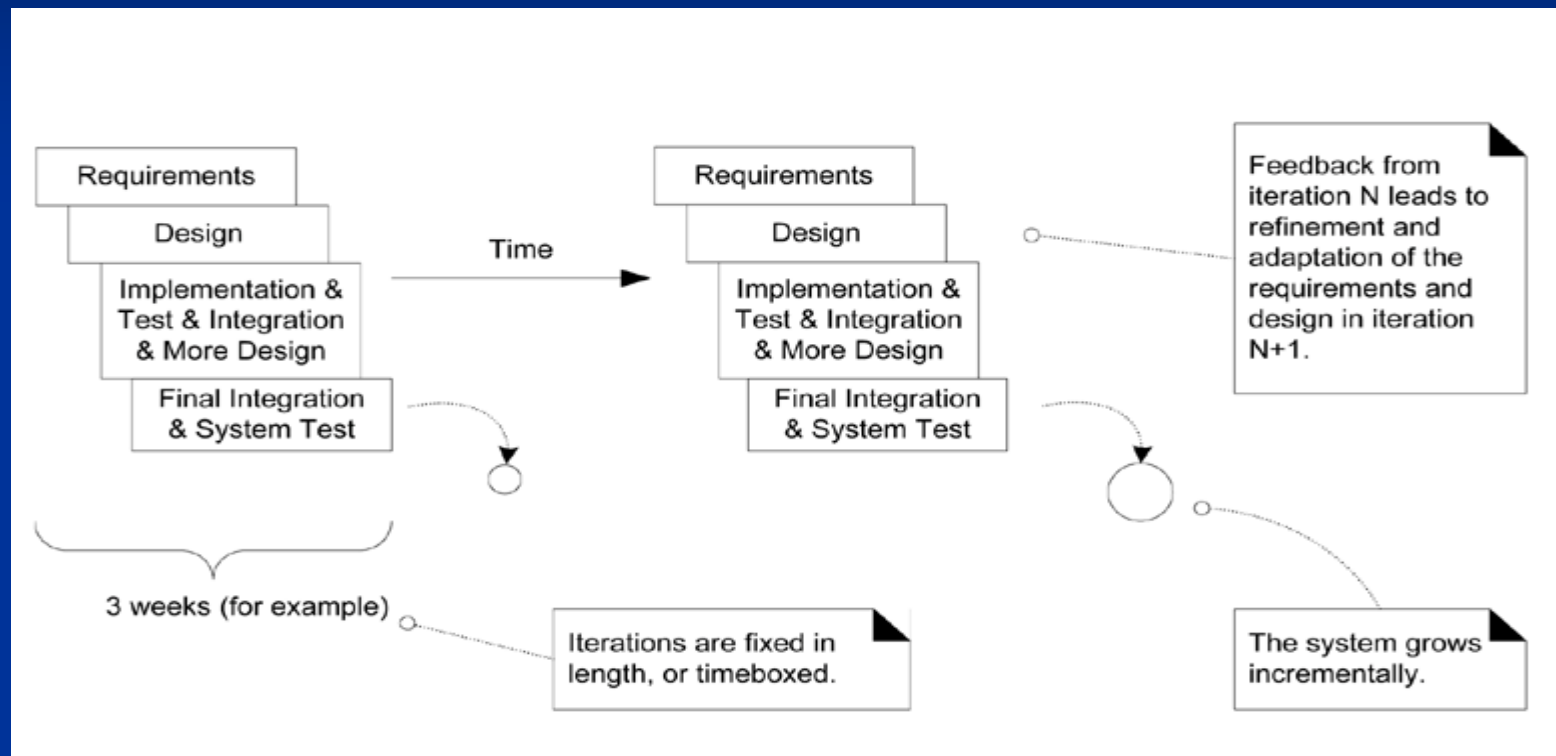
What is Iterative and Evolutionary Development?

- A key practice in UP and most other modern methods is iterative development.
- In this lifecycle approach, development is organized into a series of short, fixed-length (for example, three-week) mini-projects called iterations; the outcome of each is a tested, integrated, and executable partial system.
- Each iteration includes its own requirements analysis, design, implementation, and testing activities.

What is Iterative and Evolutionary Development?

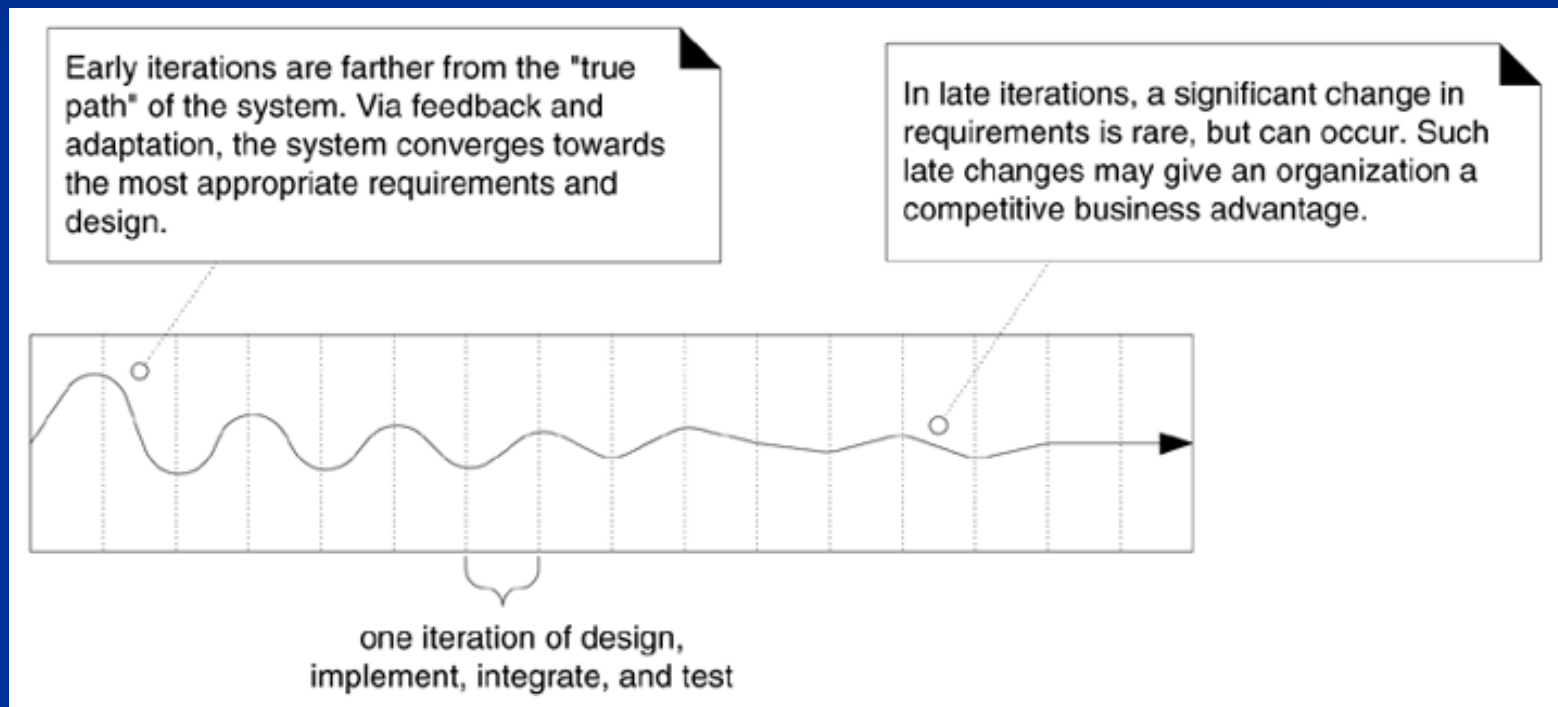
- The iterative lifecycle is based on the successive enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation as core drivers to converge upon a suitable system.
- The system grows incrementally over time, iteration by iteration, and thus this approach is also known as iterative and incremental development .
- Because feedback and adaptation evolve the specifications and design, it is also known as iterative and evolutionary development.

What is Iterative and Evolutionary Development?



Iterations lead toward the Targeted System

- Iterative feedback and evolution leads towards the desired system. The requirements and design instability lowers over time.



Benefits to Iterative Development

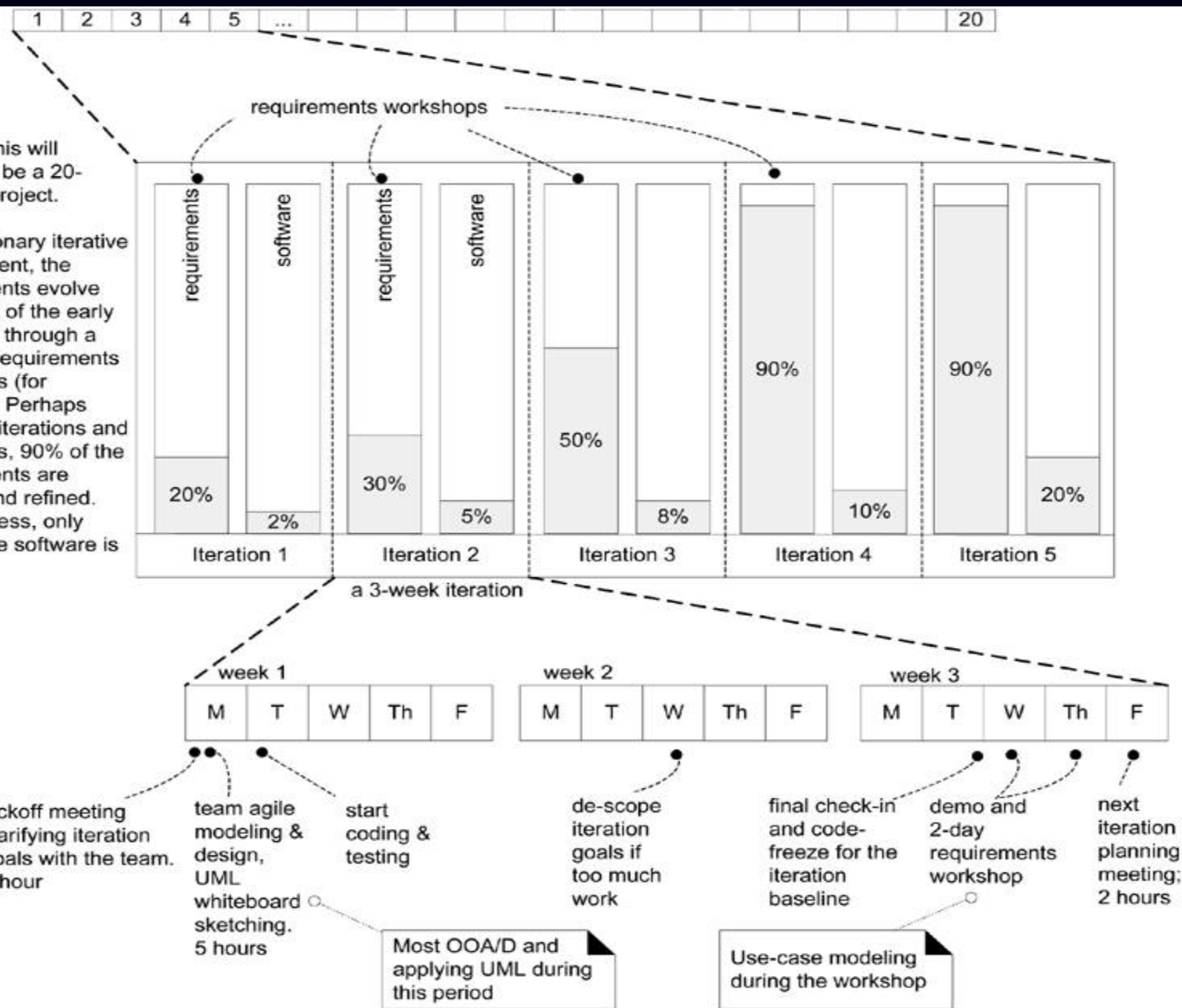
1. less project failure, better productivity, and lower defect rates; shown by research into iterative and evolutionary methods
2. early rather than late mitigation of high risks (technical, requirements, objectives, usability, and so forth)
3. early visible progress
4. early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders
5. managed complexity; the team is not overwhelmed by "analysis paralysis" or very long and complex steps
6. the learning within an iteration can be methodically used to improve the development process itself, iteration by iteration

How Long Should an Iteration Be?

- Most iterative methods recommend an iteration length between two and six weeks.
- Small steps, rapid feedback, and adaptation are central ideas in iterative development; long iterations subvert the core motivation for iterative development and increase project risk.
- A very long timeboxed iteration misses the point of iterative development. Short-iteration is good.

Imagine this will ultimately be a 20-iteration project.

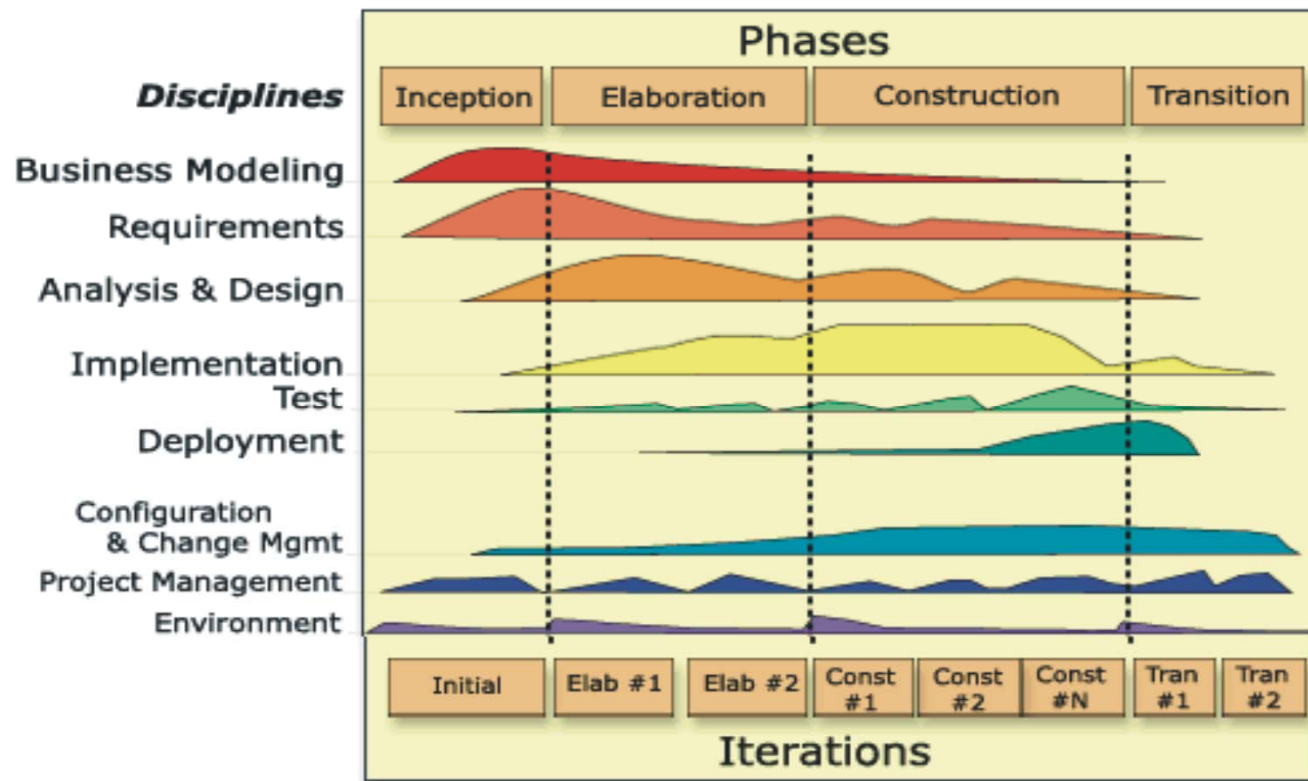
In evolutionary iterative development, the requirements evolve over a set of the early iterations, through a series of requirements workshops (for example). Perhaps after four iterations and workshops, 90% of the requirements are defined and refined. Nevertheless, only 10% of the software is built.



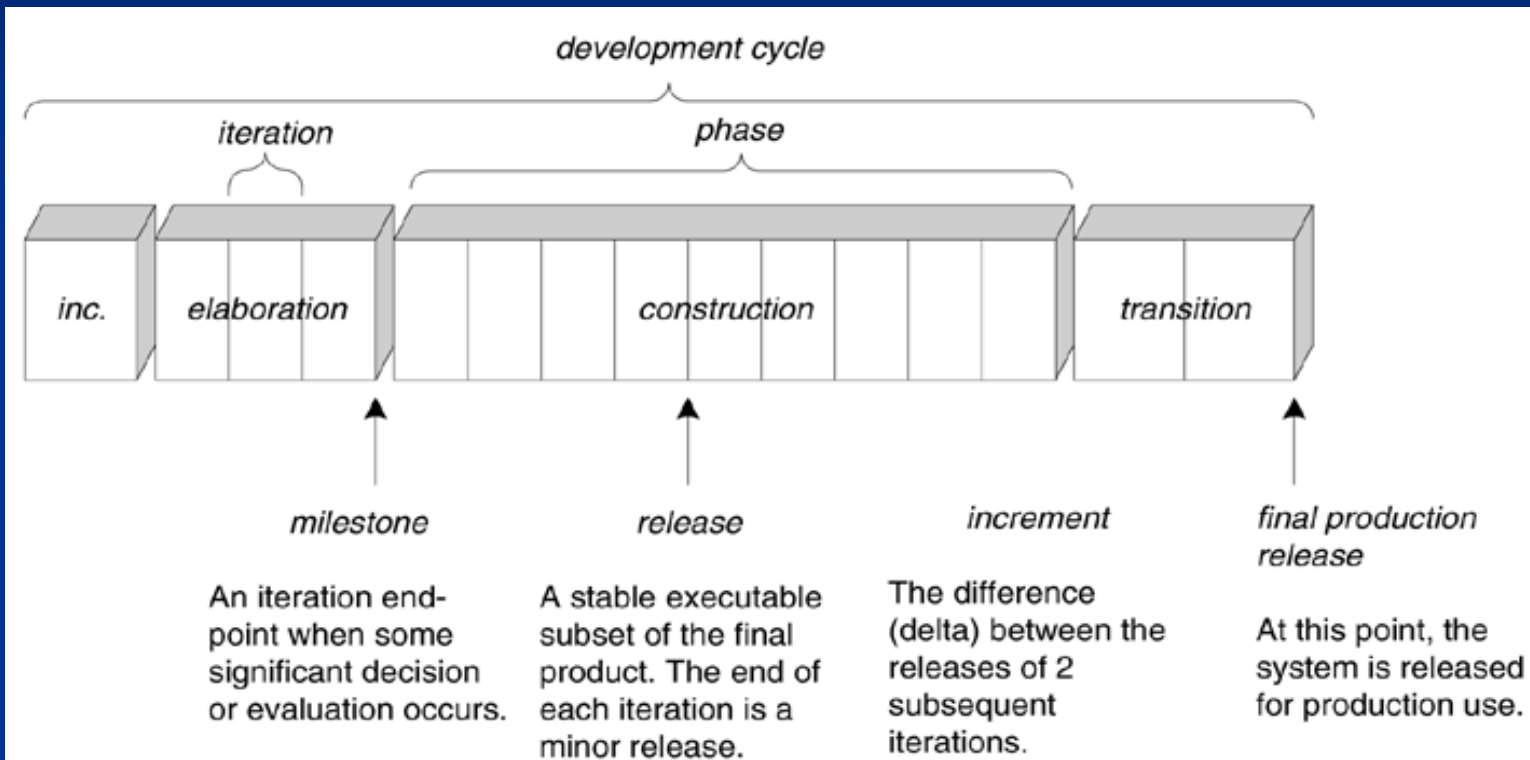
UP Phases

- A UP project organizes the work and iterations across four major phases:
 - Inception approximate vision, business case, scope, vague estimates.
 - Elaboration refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates.
 - Construction iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.
 - Transition beta tests, deployment.

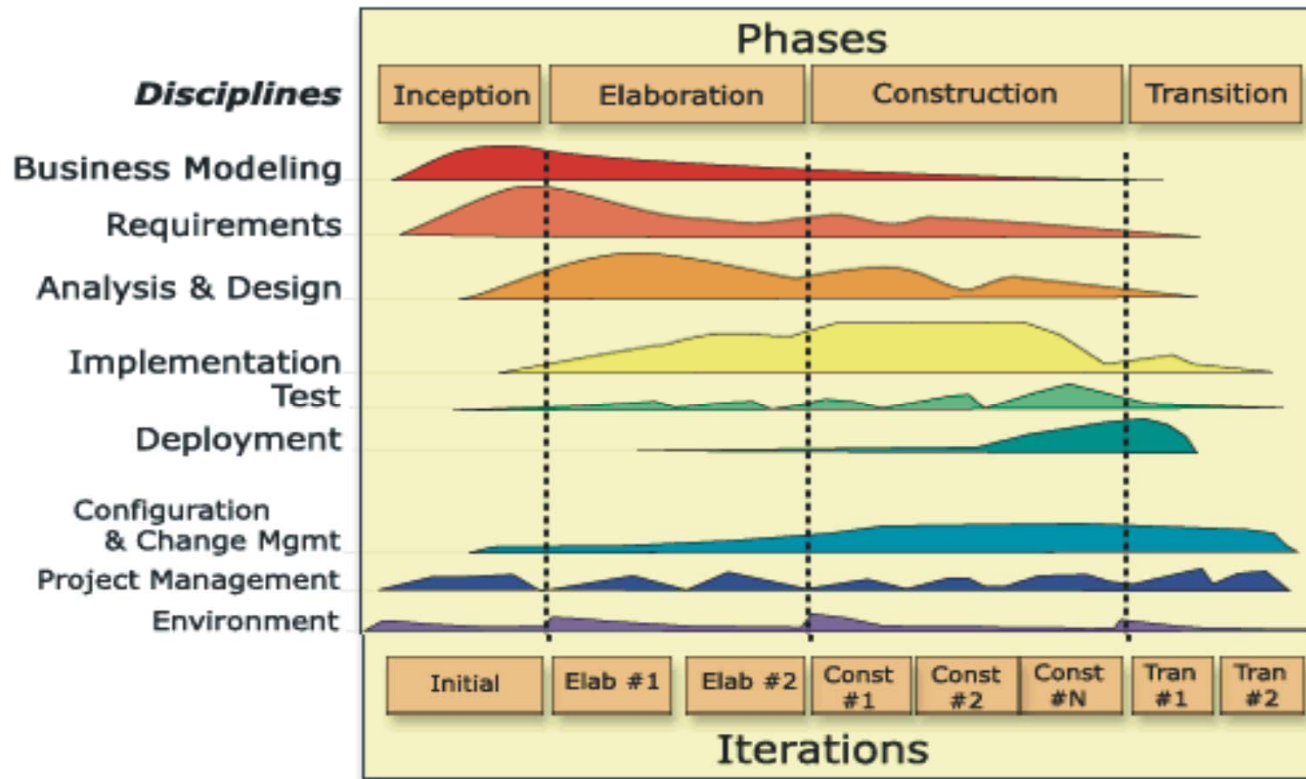
Schedule-oriented terms in the UP



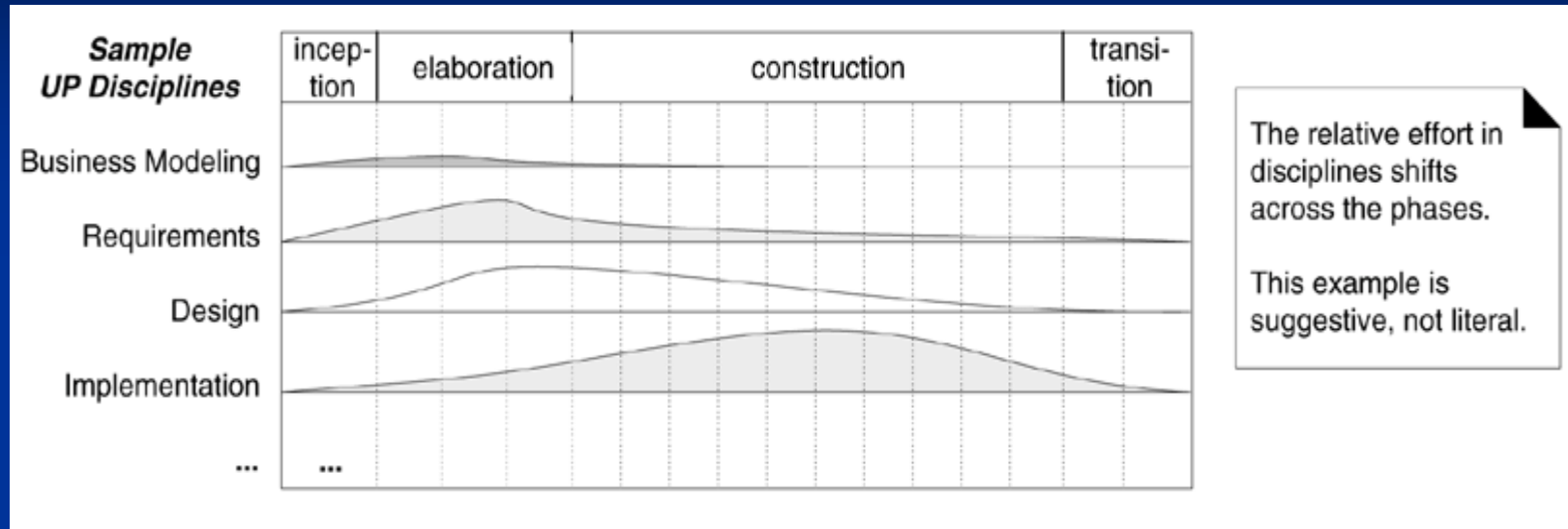
Schedule-oriented terms in the UP



UP Disciplines



Relationship Between the Disciplines and Phases?



UP Development Case

- The choice of practices and UP artifacts for a project may be written up in a short document called the Development Case (an artifact in the Environment discipline).

Discipline	Practice	Artifact	Incep.	Elab.	Const.	Trans.
		Iteration →	I1	E1..En	C1..Cn	T1..T2
Business Modeling	agile modeling req. workshop	Domain Model		s		
Requirements	req. workshop vision box exercise dot voting	Use-Case Model	s	r		
		Vision	s	r		
		Supplementary Specification	s	r		
		Glossary	s	r		
Design	agile modeling test-driven dev.	Design Model		s	r	
		SW Architecture Document		s		
		Data Model		s	r	
Implementation	test-driven dev. pair programming continuous integration coding standards	...				
Project Management	agile PM daily Scrum meeting	...				
...						

You Know You Didn't Understand Iterative Development or the UP When...

- You try to define most of the requirements before starting design or implementation. Similarly, you try to define most of the design before starting implementation; you try to fully define and commit to an architecture before iterative programming and testing.
- You spend days or weeks in UML modeling before programming, or you think UML diagramming and design activities have to fully and accurately define designs and models in great detail. And you regard programming as a simple mechanical translation of these into code.
- You think that inception = requirements, elaboration = design, and construction = implementation (that is, superimposing the waterfall on the UP).
- You think that the purpose of elaboration is to fully and carefully define models, which are translated into code during construction.

You Know You Didn't Understand Iterative Development or the UP When...

- You believe that a suitable iteration length is three months long, rather than three weeks long.
- You think that adopting the UP means to do many of the possible activities and create many documents, and you think of or experience the UP as a formal, fussy process with many steps to be followed.
- You try to plan a project in detail from start to finish; you try to speculatively predict all the iterations, and what should happen in each one.

Inception

Inception is Not the Requirements Phase

Inception is Not the Requirements Phase

- Inception is the initial short step to establish a common vision and basic scope for the project.
- It will include analysis of perhaps 10% of the use cases, analysis of the critical non-functional requirement, creation of a business case, and preparation of the development environment so that programming can start in the following elaboration phase.

What is Inception?

- Most projects require a short initial step in which the following kinds of questions are explored:
 - What is the vision and business case for this project?
 - Feasible?
 - Buy and/or build?
 - Rough unreliable range of cost: Is it \$10K–100K or in the millions?
 - Should we proceed or stop?
- Defining the vision and obtaining an order-of-magnitude (unreliable) estimate requires doing some requirements exploration.
- However, the purpose of the inception phase is not to define all the requirements, or generate a believable estimate or project plan.

Artifacts- Inception

Artifact	Comment
Vision and Business Case	Describes the high-level goals and constraints, the business case, and provides an executive summary.
Use-Case Model	Describes the functional requirements. During inception, the names of most use cases will be identified, and perhaps 10% of the use cases will be analyzed in detail.
Supplementary Specification	Describes other requirements, mostly non-functional. During inception, it is useful to have some idea of the key non-functional requirements that have will have a major impact on the architecture.
Glossary	Key domain terminology, and data dictionary.
Risk List & Risk Management Plan	Describes the risks (business, technical, resource, schedule) and ideas for their mitigation or response.
Prototypes and proof-of-concepts	To clarify the vision, and validate technical ideas.
Iteration Plan	Describes what to do in the first elaboration iteration.
Phase Plan & Software Development Plan	Low-precision guess for elaboration phase duration and effort. Tools, people, education, and other resources.
Development Case	A description of the customized UP steps and artifacts for this project. In the UP, one always customizes it for the project.

Things Go Wrong in Inception When-:

- It is more than "a few" weeks long for most projects.
- There is an attempt to define most of the requirements.
- Estimates or plans are expected to be reliable.
- You define the architecture (this should be done iteratively in elaboration).
- You believe that the proper sequence of work should be:
 - 1) define the requirements;
 - 2) design the architecture;
 - 3) implement.
- There is no Business Case or Vision artifact.
- All the use cases were written in detail.
- None of the use cases were written in detail; rather, 10–20% should be written in detail to obtain some realistic insight into the scope of the problem.

Evolutionary Requirements

Definition- Requirements

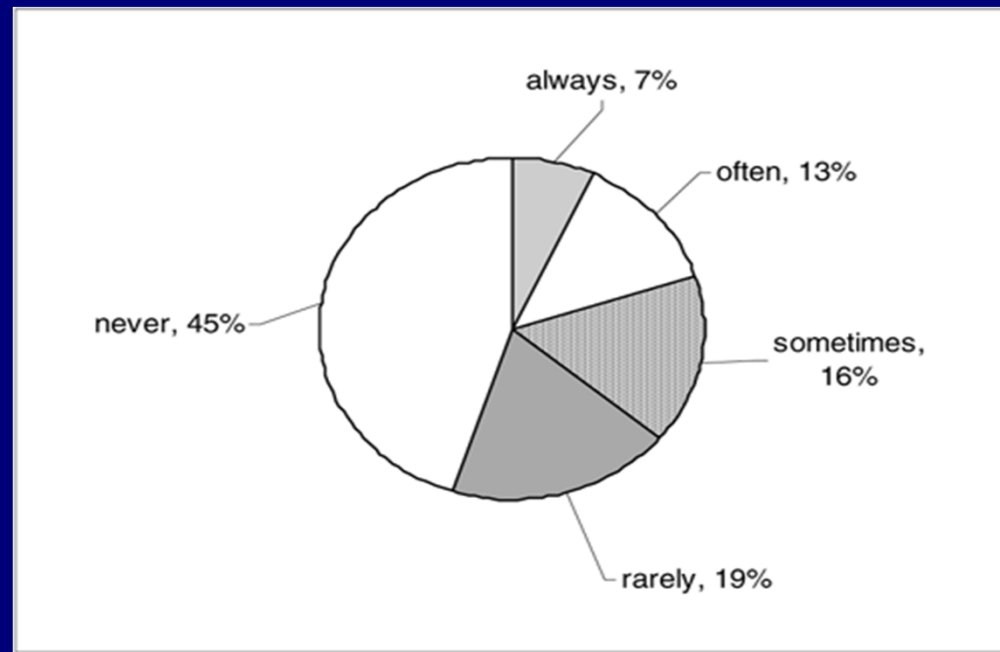
- Requirements are capabilities and conditions to which the system—and more broadly, the project—must conform
- Manage requirements- In the context of inevitably changing and unclear stakeholder's wishes, this means— "a systematic approach to finding, documenting, organizing, and tracking the *changing* requirements of a system“.

Evolutionary vs. Waterfall Requirements

- The UP embraces change in requirements as a fundamental driver on projects.
- That's important and at the heart of waterfall versus iterative and evolutionary thinking.
- Attempting waterfall practices (including detailed upfront requirements) was the single largest contributing factor for failure, being cited in 82% of the projects as the number one problem.

Evolutionary vs. Waterfall Requirements

- Shown are the result based on a research -: 45% of such early specified features were never used and an additional 19% were "rarely" used. Almost 65% of the waterfall-specified features were of little or no value!



Skillful Means to Find Requirements?

- Writing use cases with customers.
- Requirements workshops that include both developers and customers.
- Focus groups with proxy customers.
- Demo of the results of each iteration to the customers, to solicit feedback.

Types and Categories of Requirements

- In the UP, requirements are categorized according to the FURPS+ model
 - **Functional**— features, capabilities, security.
 - **Usability**— human factors, help, documentation.
 - **Reliability**— frequency of failure, recoverability, predictability.
 - **Performance**— response times, throughput, accuracy, availability, resource usage.
 - **Supportability**— adaptability, maintainability, internationalization, configurability.

Types and Categories of Requirements

- The "+" in FURPS+ indicates additional factors, such as:
 - **Implementation**— resource limitations, languages and tools, hardware.
 - **Interface**— constraints imposed by interfacing with external systems.
 - **Operations**— system management in its operational setting.
 - **Packaging**— for example, a physical box.
 - **Legal**— licensing and so forth.

Organizing Requirements

- Use-Case Model— A set of typical scenarios of using a system. There are primarily for functional (behavioral) requirements.
- Supplementary Specification—This artifact is primarily for all non-functional requirements, such as performance or licensing.
- Glossary— In its simplest form, the Glossary defines noteworthy terms. It also encompasses the concept of the data dictionary, which records requirements related to data, such as validation rules, acceptable values, and so forth.
- Vision— Summarizes high-level requirements that are elaborated in the Use-Case Model and Supplementary Specification, and summarizes the business case for the project.
- Business Rules— Business rules (also called Domain Rules) typically describe requirements or policies that transcend one software project—they are required in the domain or business, and many applications may need to conform to them. An excellent example is government tax laws.

Requirements

“What the customer wanted.”

Requirements

Requirement – a capability needed by a user to solve a problem or achieve an objective

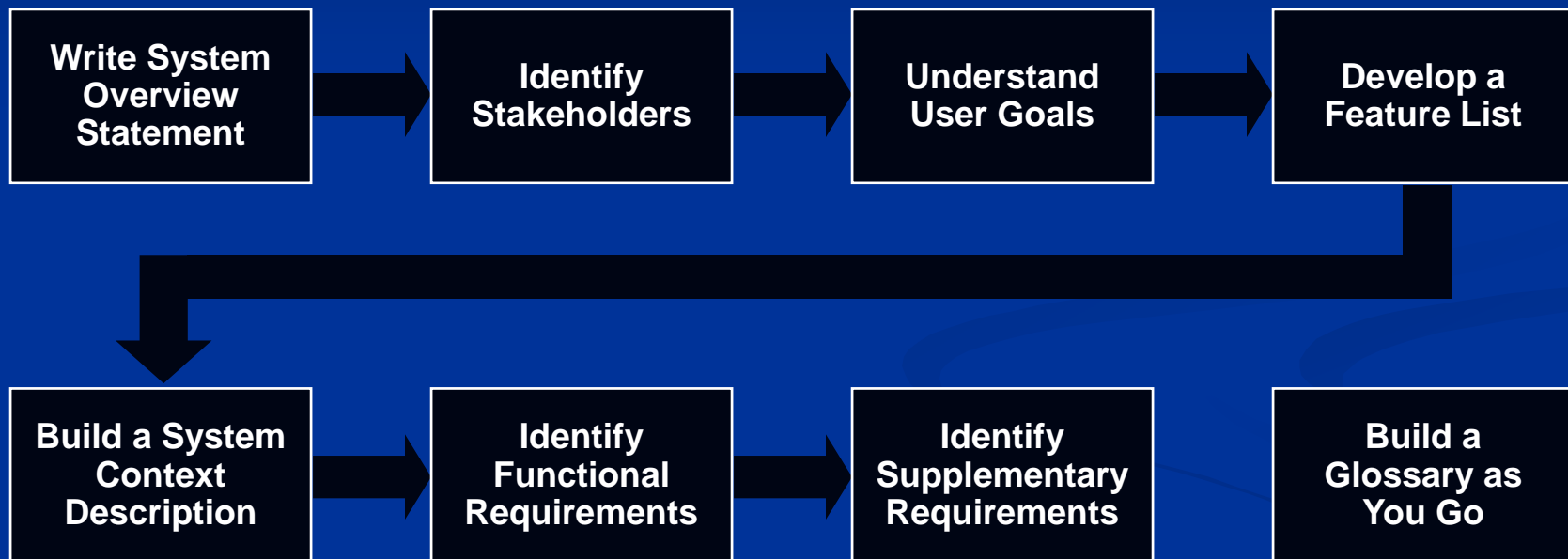
Requirements document – an unambiguous (written) description of the external behavior of the system to be built

Requirements

- Requirements Capture (A List of Requirements)
 - Delimit the system
 - Form agreement on what the system should do
 - Provide a basis for planning the technical content of iterations
 - Give developers a better understanding of the system
- Roles of requirements
 - Describe customer needs and desires for the system.
 - Provide the framework for defining the scope of the system.
 - Identify broad areas of risk, reuse, and systems-level interdependencies.
 - Establish essential contractual obligations.
 - Provide the foundation for testing criteria.
 - Requirements gathering and analysis are not specifically O-O activities

The Unified Process Requirements Workflow

Requirements workflow



Start with a Project Overview Statement

- Project overview statement should be considered an executive summary or extended abstract, giving the highest-level view of the system
- Contains all the essential project elements:
 - *Brief project description.* Defines the purpose of the project
 - (Optional) Competitive analysis.
 - *Project objective(s).* Identify concrete accomplishments and/or deliverables
 - *Examples:* 'Reduce prescription fulfillment time by 50%' or 'Provide on-line, up-to-date patient medication reports'
 - *Qualitative estimates of benefits and costs.* Include both tangibles and intangibles
 - *Examples:* 'Increase patient confidence levels' or 'Will require moderate-to-high investment in staff training'

Write *System Overview Statement*

- Contains all the essential project elements:
 - **Scope.** Indicate what capabilities will and will not be included in the project. State the boundaries of a (sub)system if it works with other (sub)systems
 - *Examples:* 'System will monitor all medications prescribed within the hospital network' or 'System will not attempt to monitor prescriptions outside the hospital network'
 - **Interfaces.** Identify other systems (computer or otherwise) with which the system must interact
 - *Examples:* Central patient records and office systems of doctors affiliated with hospital network
- Gives the broadest, highest-level picture of the system.

Identify stakeholders

- Stakeholders are those people or organizations that will be affected directly or indirectly by the system or who will have an effect on the system
- Examples of stakeholders include:
 - The system (non-end-user) customer
 - Various users of the system: end-users, system administrators, configurers, training staff, service staff
 - Development organization, individual developers
 - Marketing, sales, and distribution
 - Opponents and supporters
- We will primarily address system customers and users, focusing on the latter

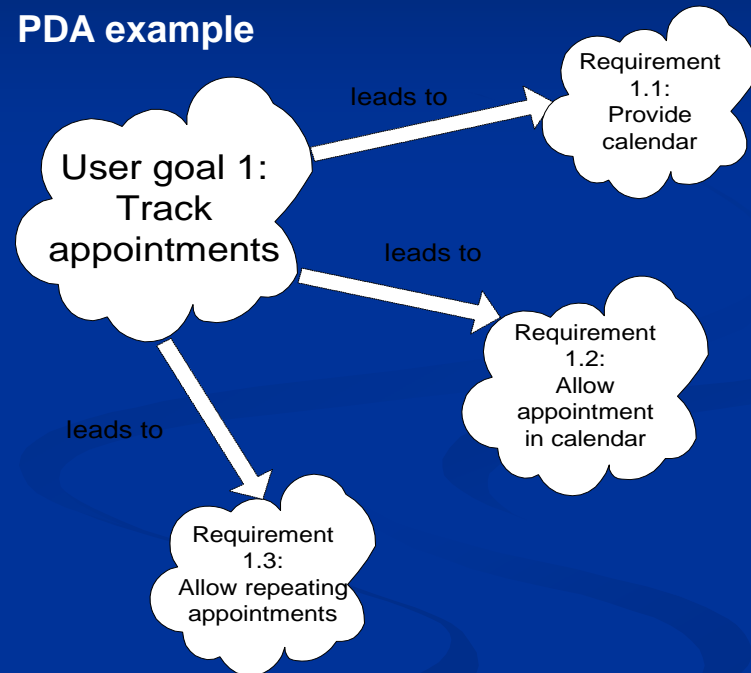
System customers and users

- System customers are those entities responsible for initiating the system project and providing material, financial, or staffing resources for system
- Actual users may be a subset of system customers or may be a completely disjoint set
- Be as specific as possible in identifying system customers and users
 - *Medical records PDA example: Customer will be the hospital or department administration; users will be doctors, nurses, physical therapists, etc.*
- May need to reconcile system customer goals and user goals

Understand user goals

- User goals are specific, but not detailed, descriptions of what user wants the system to do.
- A user goal may lead to one or more requirements.
- User goals provide the foundation for use-case analysis.

PDA example



Develop a feature list

- Feature list identifies a list of *high- or mid-level* requirements, out of a (usually) much larger wish list, that are *candidates* for inclusion in the system.
- For each requirement, provide:
 - Number each feature, e.g. F003
 - A name, as descriptive as possible.
 - Description in the form of an explanation or a definition.
 - Associated user goal (*optional*). A single goal may have more than one requirement associated with it.
 - Priority (*required, desirable, or optional*).
 - Visibility (*visible to user or hidden from user*).
 - Estimated *implementation* risk for this requirement (*high, normal, minimal*).

Build a system context description

- *System context description* describes the environment in which the system is embedded.
- Identifies entities that are directly relevant to the system:
 - Domain objects.
 - Domain processes.
 - Domain actors.
- These are examples of *conceptual classes*
- System context description acts as the foundation for the *Domain Model*.

System context: identify domain objects

- *Domain objects* are the entities encountered in the world in which the system will operate.
- Domain objects for (Medical) PDA software include:
 - *'Business' objects and abstractions:* Patient record, appointment record, medication list, synchronization schedule.
 - *Real-world objects:* Touch screen, synchronization link (e.g. cradle, IR, Bluetooth), host PC
 - *Events:* Last synchronization date/time with host, create new medication record request, medication alarm notification; anything that occurs within a limited time

System context: identify domain processes

- Discovery of *domain processes* often proceeds concurrently with domain object discovery.
- Types of domain processes might include:
 - *Start-up*: Touch screen initialization; set time, time zone, and date.
 - *Standard operating procedures*: Add/delete PDA database entry, change font size, soft reset.
 - *Exceptional circumstances*: 'Hard' reset, delete private records.
- For requirements phase, just name and define processes.
- *Do not explore process detail here!* This will be captured later in use-case descriptions, sequence diagrams, and statecharts.

System context: identify domain actors

- *Domain actors* interact in some way with the system
 - Users are a subset of domain actors
 - Actors may be *active* (directly interacting with system) or *passive* (indirect interaction)
 - Actors may be human or non-human
- *Examples:*
 - Active/human: device user (doctor, nurse, etc.)
 - Active/non-human: PDA/host synchronization software
 - Passive/human: recipient of consultation reminder
 - Passive/non-human: PC-based calendar/contact manager (e.g. MS Outlook)

Build a glossary as you go

- *Glossary* is the repository for any terms used to describe the system.
- Includes items identified in the *System Context Description* as well as any terms encountered in interfaces to the system.
- Include only terms that are unique or the definitions of which differ from their general use.
- Will evolve into the standard reference for abstractions.
- Benefits:
 - Establishes a common and consistent vocabulary among all stakeholders
 - Provides global view which may lead to discovering commonalties and simplification.
 - Allows browsing which promotes reuse.

Identify functional requirements

- Features vs. functional requirements:
 - A **feature** may have behavior or it may not, e.g. 'Use a color display' or 'support non-Latin character sets.'
 - A **functional requirement** has behavior, e.g. 'Sound alarm n minutes before appointment.'
 - Functional requirements represent a subset of the feature list.
- Functional requirements specify *what* behavior the system will exhibit, but not *how* the behavior will be implemented!
- Some functional requirements can be identified from the system features list and the system context description.
- Most functional requirements are discovered through *use cases*.

Identify functional requirements

- *Note:* Functional requirements are only one category within FURPS+, but represent most of the requirements effort.
- Functional requirements represent the features and capabilities of the system: the 'F' in FURPS+
- *Examples:*
 - 'Allow real-time update of patient medication schedule'
 - 'Notify user of drug interaction warnings when new medication prescribed'
 - 'Provide means for scheduling consultations'

Identify supplementary requirements

- *Supplementary requirements* are those not associated with a particular functional requirement of the system.
- May span several functional requirements, use cases, or none at all.
- Some may be addressed by aspect-oriented programming (AOP), e.g. error handling and logging, failure recovery
- More examples:
 - Hardware platforms, system software, **security**, reliability and **availability**, **performance**, file formats.
 - System **modifiability**, subsystem availability, ease of learning.
 - Legal, regulatory, fiduciary responsibility, and liability requirements or constraints.

Feature/requirement example

- Consider the requirement to monitor all medications prescribed within the hospital network

Name: Monitor all prescribed medications

Description: Monitor all medications prescribed within the hospital network. System will not monitor prescriptions from outside the hospital network

User goals: Avoid unwanted drug interactions; avoid over- or under-medicating patients

Priority: Required

Visibility: Visible

Implementation risk: Normal

Conceptual class categories

- *Conceptual class categories* are a way of helping identify domain objects, processes, and actors
- These conceptual class categories are adapted from *Applying UML and Patterns*, Second & Third Editions, by Craig Larman

Conceptual Class Category	Example
Physical or tangible objects	TouchScreen, PDA, HostPC
Specifications, designs, or descriptions	Consultation-Protocol
Places	PatientBed
Transactions	DrugDosage-Change
Transaction line items	OriginalDose, NewDose
Roles of people	Patient, Primary-CarePhysician
Containers	RadiologyLab, OperatingRoom
Contained items	PatientMonitor, MRIUnit

Conceptual class categories

Conceptual Class Category	Example
Other collaborating systems	Originating hospital, primary care physician's records
Abstract nouns concepts	QualityOfLife
Organizations	AMA, InsuranceCompany
Events	PatientAdmitDate, PatientDischargeDate
Processes	MedicalHistoryReview, PatientAdmission
Rules and policies	DischargePolicy
Catalogs	AllowedProcedures
Records of finance, work, contracts, legal matters	Receipt, PreAuthorization
Financial instruments and services	PatientAccount, PatientCharge
Manuals, schedules, documents, references	PhysiciansDeskReference

Requirements

A *Requirements document* can include:

- Overview statement (purpose of the project)
 - Charter
- Vision statement
- Scope: what is covered and what is not
- Customers (who will use the product / buy the product)
- Goals (support faster, cheaper, more reliable, etc. Business operations)
- System functions (what the system is supposed to do)
- System attributes (ease of use, fault tolerance, response time, etc.)
- Competitive analysis: who is doing something similar and what does the product do (features)
- System Context: description of major processes, actors and concepts

Requirements

Requirements should:

- Be measurable
 - Not “fast”, but “10ms to accomplish . . .”
- Have verifiable objectives
 - Not “easy to use”
- Have concrete requirements
- Must be able to determine if requirement has been satisfied by the design.
- Must be traceable

Requirements

Requirements document should not include:

- Discussion of the language to implement
- Details on how the system should work
- Details on the equipment involved
- Algorithms or implementation details

Typical problem with many requirement statements:

- Embedded design (How vs. What).
- Vague
- Computer industry language instead of user's language.

Business Rules

Business Rules

Business Rules – dictate how a company conducts its business.

Business rules are likely to apply to several or all use cases developed for a particular application.

Domain Rules – facts and information on the domain.

Domain rules are likely to apply to several or all use cases developed for a particular application.

Business Rules

- Rules and policy required by the “business” or other agency (IRS)
 - Checks must be approved by manager
 - Credit cards must get approval
 - Books are due in 14 days
 - Tax must be collected for all sales
 - 14 day wait for purchasing firearms
 - No cigarettes to minors

Business Rules

Five Categories of Business Rules

- Structural facts -- Facts or conditions that must be true.
 - Example: The first customer contact is always with a salesperson.
- Action restricting -- Prohibiting one or more actions based on a condition.
 - Example: Do not accept a check from a customer who does not have an acceptable credit history
- Action triggering -- Starts action when one or more conditions become true.
 - Example: Send the shipment as soon as the pick items have been collected
- Inferences -- Conclusions formed when one or more conditions become true.
 - Example: Members who fly more than 100,000 miles a year become Elite members.

Business Rules

Five Categories of Business Rules

- Calculations -- Calculating one value from a set of other values.
 - Example: The sales amount is the total retail value of the line items but does not include state or federal tax.

Business Rules

Business Rule Catalog – specified at the finest level of granularity possible

No.	Rule Definition	Changeable	Source
BR001	Cash, personal check, and credit card are accepted for rental payment	Yes	Management policy
BR002	Customers who rent driving equipment must possess a driver's license.	No	Management policy
BR003	A background check must be initiated for customers renting firearms and a 30-day waiting period is required.	Yes, law may change	Federal law

Domain Rules

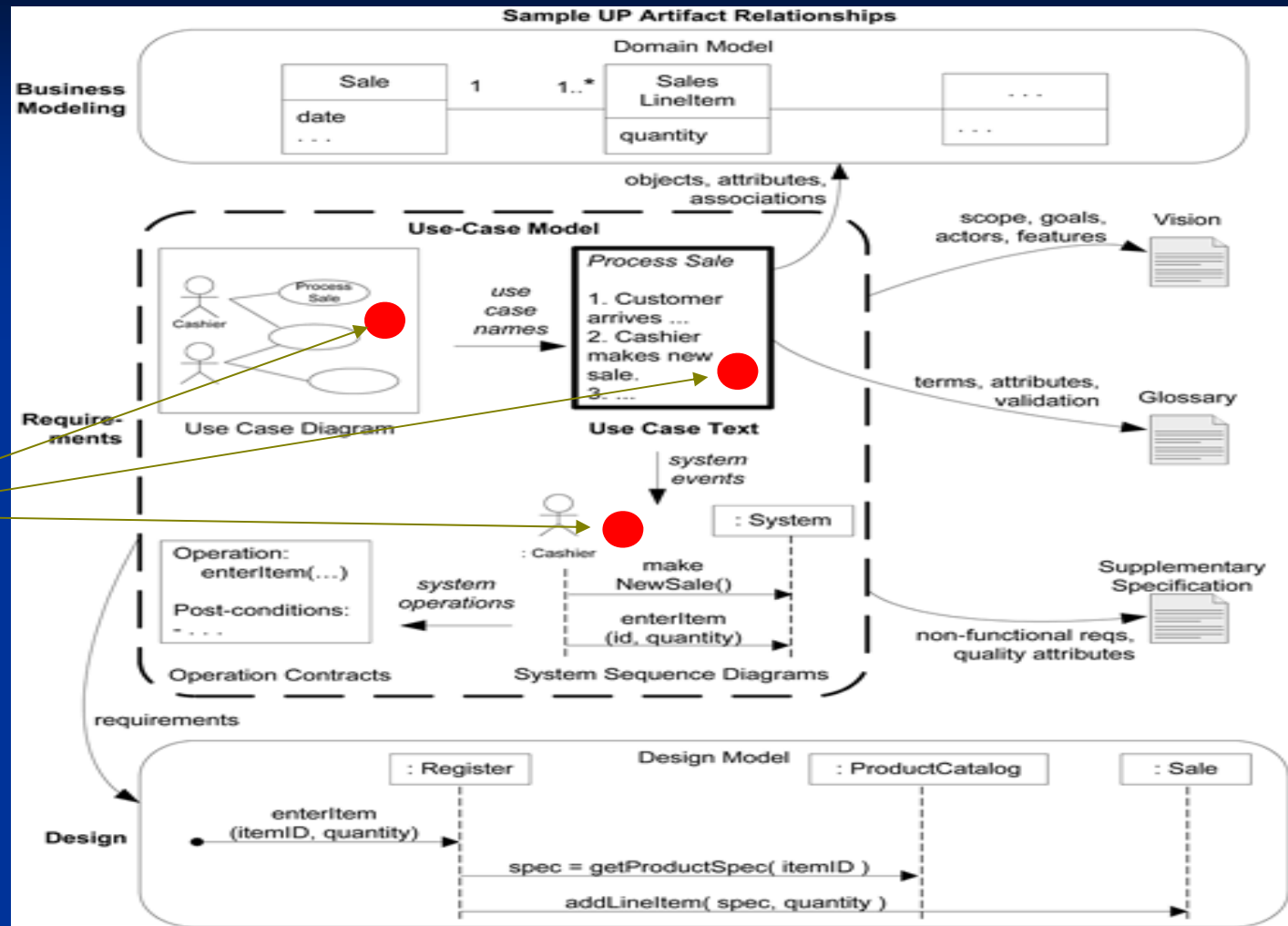
- Examples:
 - GPS has an accuracy of X meters
 - Temperature and humidity need to remain constant or within valid range specified

Use Cases

We have discussed these ...

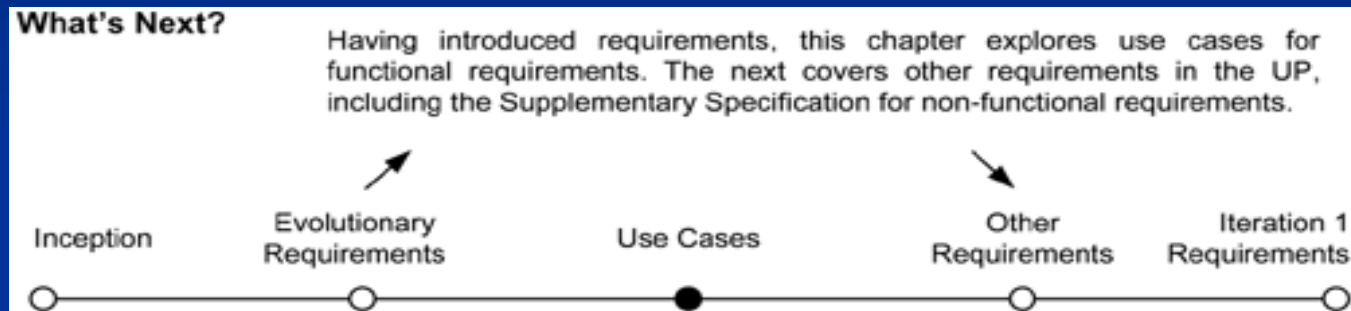


Now we will discuss these ...



Introduction

- Use cases are text stories, widely used to discover and record requirements.

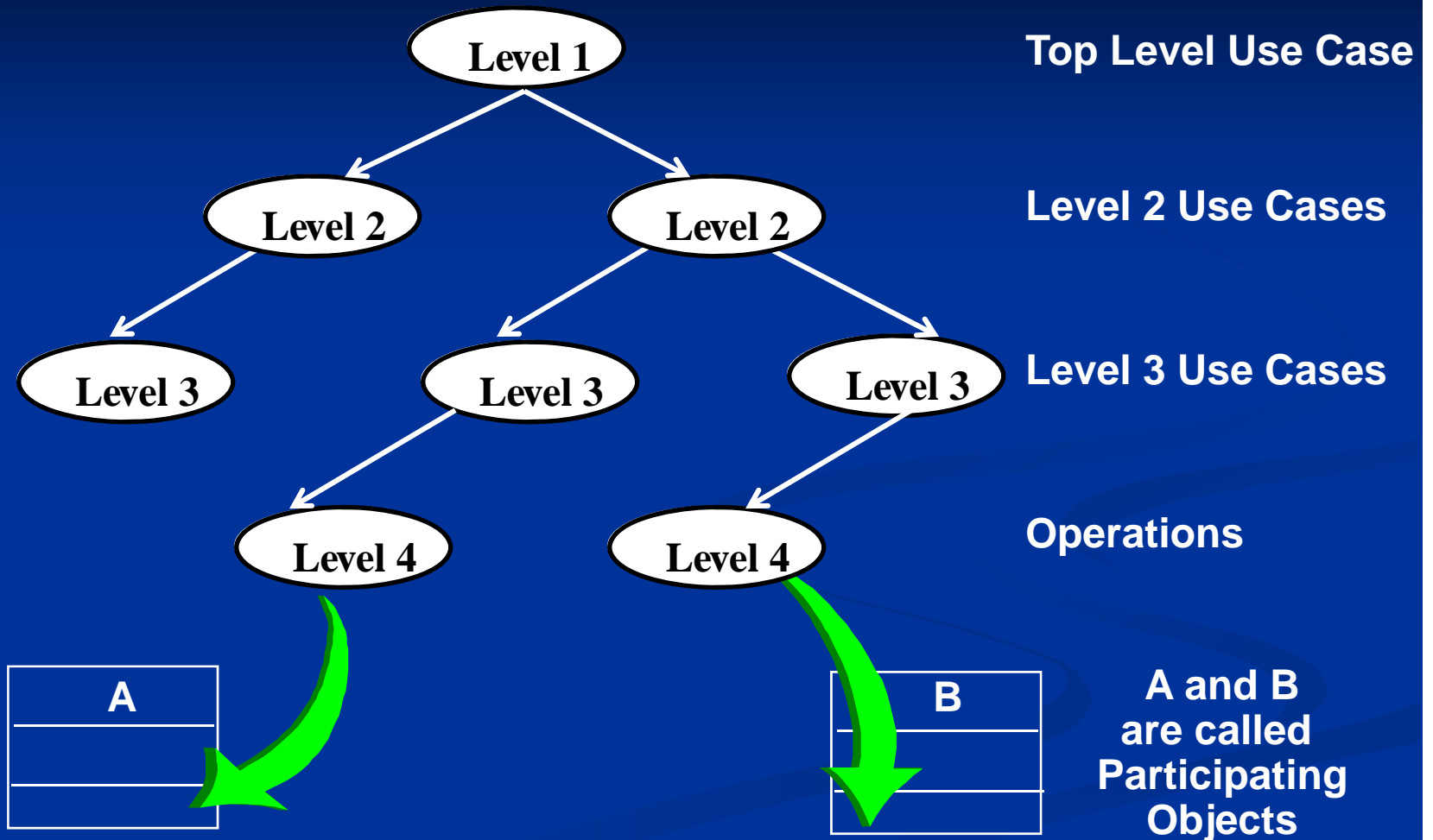


- Use cases *are* requirements, primarily functional or behavioral requirements that indicate what the system will do. In terms of the FURPS+ requirements types, they emphasize the "F" (functional or behavioral), but can also be used for other types, especially when those other types strongly relate to a use case.

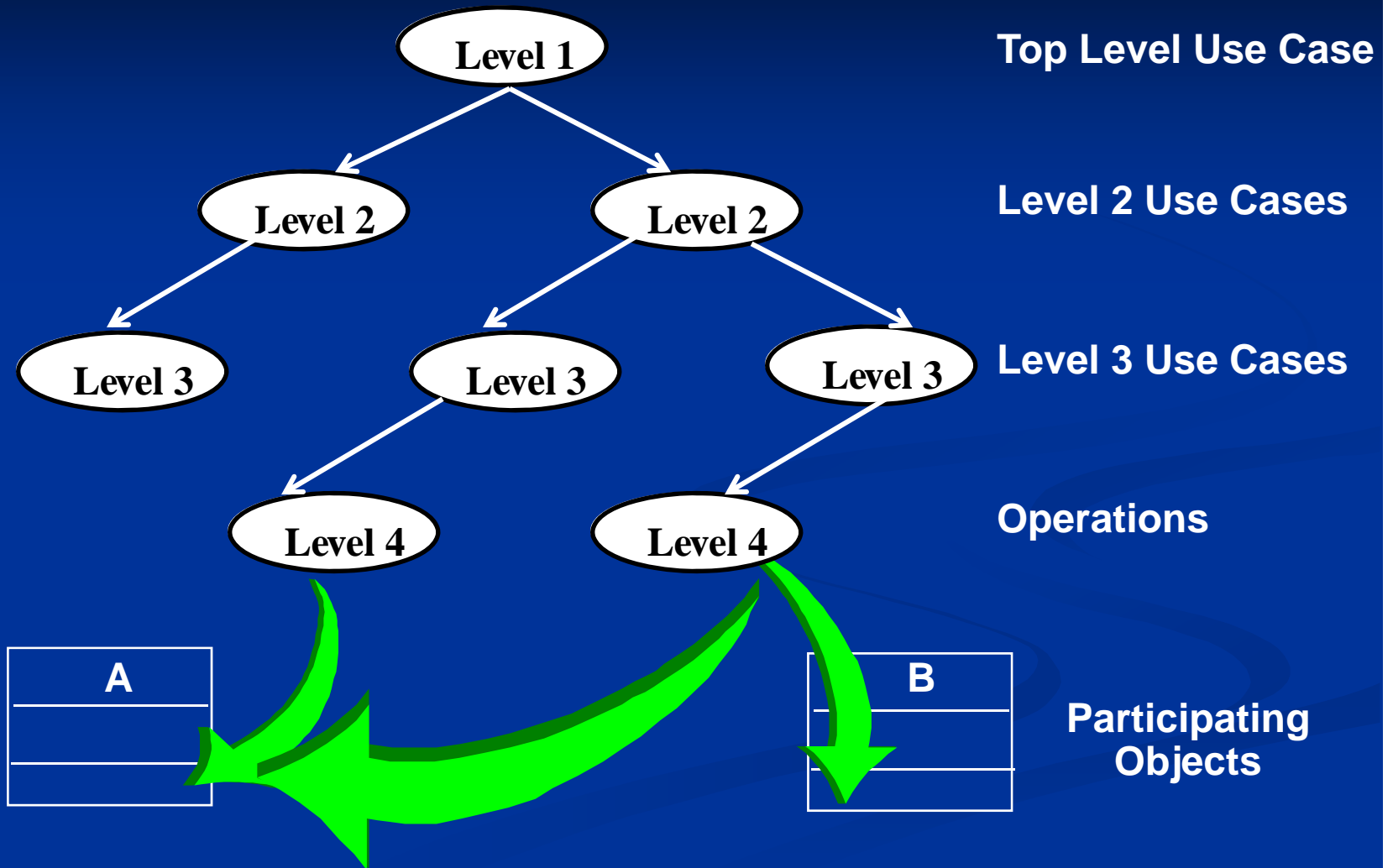
Introduction

- High-level goals and use case diagrams are input to the creation of the use case text.
- The use cases can in turn influence many other analysis, design, implementation, project management, and test artifacts.

From Use Cases to Objects

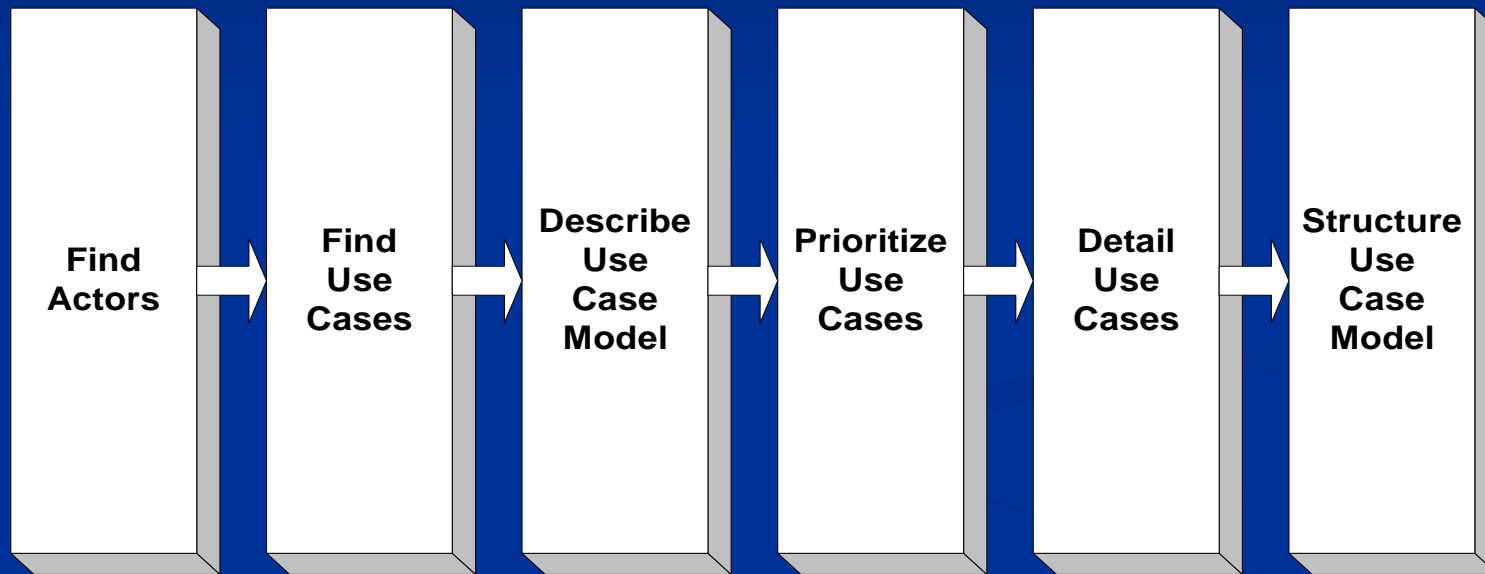


Use Cases can be used by more than one object

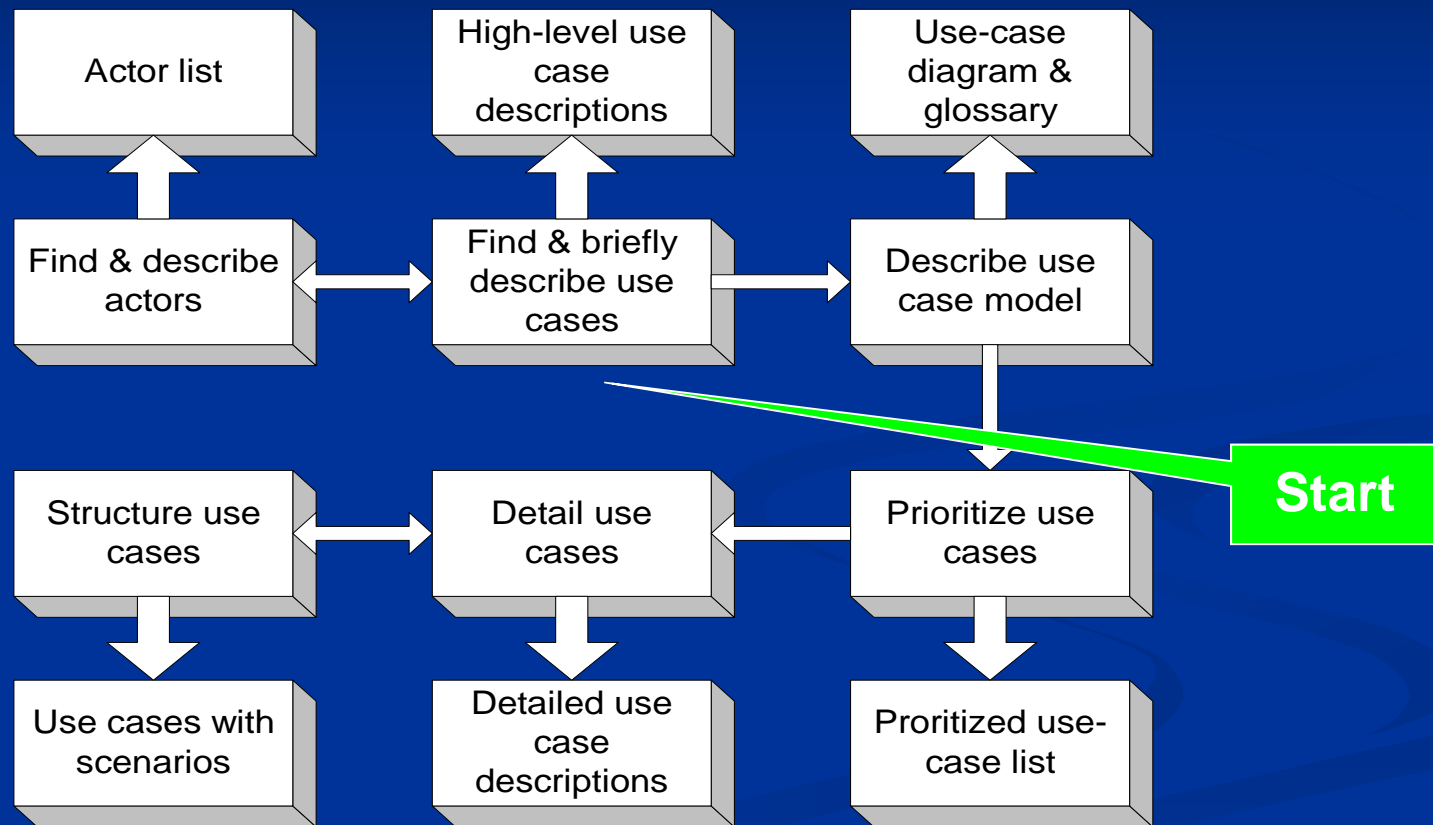


Use Case Workflow

Idealized use case workflow



UC workflow techniques and artifacts



Modeling system behavior

Typical ways of modeling system behavior

- Scenarios or story boards
- Use cases
- (System) Sequence diagram: way of drawing a picture of a scenario
- (Object) Interaction diagrams
 - Message sequence charts
 - Communication (Collaboration) Diagrams

Use cases and scenarios

- The use case describes a set of interactions between the user and the system, possibly with several different outcomes.
- A scenario describes a specific path through, and outcome of, a use case.
- A use case represents a collection of scenarios: primary, plus zero or more alternates.
- The primary scenario corresponds to the main system interactions, usually the 'success' scenario.
- Alternate scenarios correspond to less frequent interactions and exceptions.
- Different scenarios are analogous to alternatives in *switch..case* constructs.
- The term interaction can refer to a single interaction or a set. Typically an actor has a set of interactions with the system.

Scenarios

- A scenario is a little story
 - ▶ it is an outline of some expected sequence of events
- A scenario is used to convey the fundamentals of “how things work”
- It usually includes at least one actor
- Each actor can make requests of the system or respond to system activity

I would like a book of stamps, please.

OK. Will that be all?

Yes.

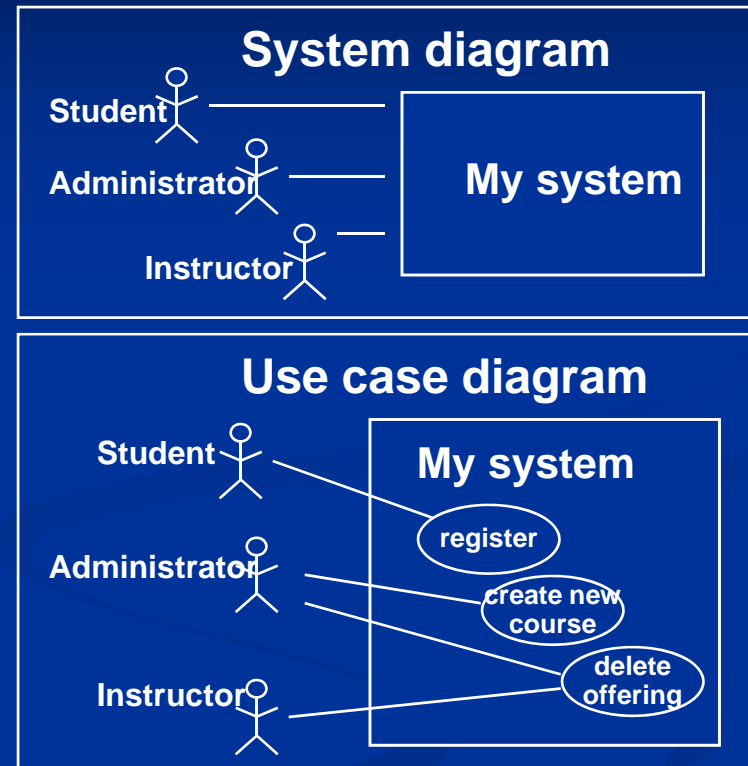
That will be \$7.80.

Here is \$10.

Thanks. Here are your stamps and your change.

High-level system view

- If we look at the system as a “black box”, we can identify some of the external users of the system (either humans or other computer systems)
 - The simplest “black box” diagram is a *system diagram*, which shows the outside actors
 - The *use case diagram* is more elaborate: it also shows the connections between “use cases” and actors



Sequence diagram

- A sequence diagram is a way of drawing a picture of a scenario
- Sequence diagrams are also sometimes called event trace diagrams, ladder diagrams, interaction diagrams, or fence post diagrams
 - Each vertical line describes an “actor” or a “system” in the scenario
 - The vertical axis represents time: time flows down the page



Iterative Development

Use Cases

- Decide and describe the functional requirements of the system
- Bring agreement between the customer and software developer
- Give a clear and consistent description of what the system should do.
- Provide a basis for performing tests that verify the system delivers the functionality stated.
- Trace the functional requirements into actual classes and operations in the system.

Why Use Cases

- Use cases are a good way to help keep it simple, and make it possible for domain experts or requirement owner to themselves write (or participate in writing) use cases.
- Another value of use cases is that *they emphasize the user goals and perspective*.
- Another motivation is to replace detailed, low-level function lists (which were common in 1970s traditional requirements methods) with use cases.
- This provided a unifying and understandable way to pull the requirements together—into stories of requirements in context of use.

Three Common Use Case Formats

- Brief— one-paragraph summary, usually of the main success scenario.
- Casual— Informal paragraph format. Multiple paragraphs that cover various scenarios.
- Fully dressed— All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.

What do various Sections Means

■ Section of Fully Dressed Use-Case Template

Use Case Section	Comment
Use Case Name	Start with a verb.
Scope	The system under design.
Level	"user-goal" or "subfunction"
Primary Actor	Calls on the system to deliver its services.
Stakeholders and Interests	Who cares about this use case, and what do they want?
Preconditions	What must be true on start, <i>and</i> worth telling the reader?
Success Guarantee	What must be true on successful completion, <i>and</i> worth telling the reader.
Main Success Scenario	A typical, unconditional happy path scenario of success.
Extensions	Alternate scenarios of success or failure.
Special Requirements	Related non-functional requirements.
Technology and Data Variations List	Varying I/O methods and data formats.
Frequency of Occurrence	Influences investigation, testing, and timing of implementation.
Miscellaneous	Such as open issues.

Definition – Use Case Sections

- Scope
 - The scope bounds the system (or systems) under design.
 - Typically, a use case describes use of one software (or hardware plus software) system; in this case it is known as a system use case.
 - At a broader scope, use cases can also describe how a business is used by its customers and partners.

Definition – Use Case Sections

- Level

1. A **user-goal level** use case is the common kind that describe the scenarios to fulfill the goals of a primary actor to get work done; it roughly corresponds to an elementary business process (EBP) in business process engineering.
2. A **sub function-level** use case describes sub steps required to support a user goal, and is usually created to factor out duplicate sub steps shared by several regular use cases (to avoid duplicating common text).

Definition – Use Case Sections

An Actor is something with behavior, such as a person identified by role), computer system, or organization; for example, a cashier.

- Primary actor— has user goals fulfilled through using services of the SuD (System under Discussion). For example, the cashier.
- Supporting actor— provides a service (for example, information) to the SuD. The automated payment authorization service is an example. Often a computer system, but could be an organization or person.
- Offstage actor— has an interest in the behavior of the use case, but is not primary or supporting; for example, a government tax agency.

Definition – Use Case Sections

- Stakeholders and Interests List

The use case, as the contract for behavior, captures *all and only* the behaviors related to satisfying the stakeholders' interests.

- Preconditions state what *must always* be true before a scenario is begun in the use case. Preconditions are *not* tested within the use case; rather, they are conditions that are assumed to be true. Typically, a precondition implies a scenario of another use case, such as logging in, that has successfully completed.

- Success guarantees (or post conditions) state what must be true on successful completion of the use case—either the main success scenario or some alternate path. The guarantee should meet the needs of all stakeholders.

Definition – Use Case Sections

Main Success Scenario (Basic Flow)

- This has also been called the "happy path" scenario, "Basic Flow" or "Typical Flow."
- It describes a typical success path that satisfies the interests of the stakeholders.
- It often does *not* include any conditions or branching. Although not wrong or illegal
- It is arguably more comprehensible and extendible to be very consistent and defer all conditional handling to the Extensions section.

Definition – Use Case Sections

Extensions (or Alternate Flows)

- Extensions are important and normally comprise the majority of the text. Sometimes, a use case branches has to perform another use case scenario.

Special Requirements

- If a non-functional requirement, quality attribute, or constraint relates specifically to a use case, record it with the use case. These include qualities such as performance, reliability, and usability, and design constraints (often in I/O devices) that have been mandated or considered likely.

Definition – Use Case Sections

- Technology and Data Variations List

Often there are technical variations in *how* something must be done, but not what, and it is noteworthy to record this in the use case. A common example is a technical constraint imposed by a stakeholder regarding input or output technologies.

Example – Use Case

- NextGen point-of-sale (POS) system



- A POS system is a computerized application used (in part) to record sales and handle payments; it is typically used in a retail store.
- It includes hardware components such as a computer and bar code scanner, and software to run the system.
- It interfaces to various service applications, such as a third-party tax calculator and inventory control.

Example – Use Case

■ Process Sale:

- A customer arrives at a checkout with items to purchase.
- The cashier uses the POS system to record each purchased item.
- The system presents running total and line-item details.
- The customer enters payment information, which the system validates and records.
- The system updates inventory.
- The customer receives a receipt from the system and then leaves with the items.

Example – Use Case

Scope: NextGen POS application

Level: user goal

Primary Actor: Cashier

Stakeholders and Interests:

- **Cashier:** Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.
- **Salesperson:** Wants sales commissions updated.
- **Customer:** Wants purchase and fast service with minimal effort. Wants easily visible display of entered items and prices. Wants proof of purchase to support returns.
- **Company:** Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.
- **Manager:** Wants to be able to quickly perform override operations, and easily debug Cashier problems.
- **Government Tax Agencies:** Want to collect tax from every sale. May be multiple agencies, such as national, state, and county.
- **Payment Authorization Service:** Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.

Example – Use Case

- Preconditions: Cashier is identified and authenticated.
- Success Guarantee (or Post conditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.
- Main Success Scenario (or Basic Flow):
 1. Customer arrives at POS checkout with goods and/or services to purchase.
 2. Cashier starts a new sale.
 3. Cashier enters item identifier.
 4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.
Cashier repeats steps 3-4 until indicates done.
 5. System presents total with taxes calculated.
 6. Cashier tells Customer the total, and asks for payment.
 7. Customer pays and System handles payment.
 8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
 9. System presents receipt.
 10. Customer leaves with receipt and goods (if any).

Example – Use Case

Please see your text book for complete list of the extensions – use cases
The following is only an example to illustrate key concepts discussed

Extensions (or Alternative Flows):

- a - At any time, Manager requests an override operation:
 - 1. System enters Manager-authorized mode.
 - 2. Manager or Cashier performs one Manager-mode operation. e.g., cash balance change, resume a suspended sale on another register, void a sale, etc.
 - 3. System reverts to Cashier-authorized mode.
- b - At any time, System fails:

To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.

 - 1. Cashier restarts System, logs in, and requests recovery of prior state.
 - 2. System reconstructs prior state.
 - 2a. System detects anomalies preventing recovery:
 - 1. System signals error to the Cashier, records the error, and enters a clean state.
 - 2. Cashier starts a new sale.

Example – Use Case

3a. Invalid item ID (not found in system):

1. System signals error and rejects entry.

2. Cashier responds to the error:

- 2a. There is a human-readable item ID (e.g., a numeric UPC):

1. Cashier manually enters the item ID.

2. System displays description and price.

- 2a. Invalid item ID: System signals error. Cashier tries alternate method.

- 2b. There is no item ID, but there is a price on the tag:

1. Cashier asks Manager to perform an override operation.

2. Managers performs override.

3. Cashier indicates manual price entry, enters price, and requests standard taxation for this amount (because there is no product information, the tax engine can't otherwise deduce how to tax it)

- 2c. Cashier performs Find Product Help to obtain true item ID and price.

- 2d. Otherwise, Cashier asks an employee for the true item ID or price, and does either manual ID or manual price entry (see above).

Example – Use Case

■ Special Requirements:

Touch screen UI on a large flat panel monitor. Text must be visible from 1 meter.

- **Credit authorization response within 30 seconds 90% of the time.**
- **Somehow, we want robust recovery when access to remote services such the inventory system is failing.**
- **Language internationalization on the text displayed.**
- **Pluggable business rules to be insert able at steps 3 and 7.**

■ Technology and Data Variations List:

Manager override entered by swiping an override card through a card reader, or entering an authorization code via the keyboard.

3a. Item identifier entered by bar code laser scanner (if bar code is present) or keyboard.

3b. Item identifier may be any UPC, EAN, JAN, or SKU coding scheme.

7a. Credit account information entered by card reader or keyboard.

7b. Credit payment signature captured on paper receipt. But within two years, we predict many customers will want digital signature capture.

■ Frequency of Occurrence: Could be nearly continuous.

■ Open Issues:

- **What are the tax law variations?**
- **Explore the remote service recovery issue.**
- **What customization is needed for different businesses?**
- **Must a cashier take their cash drawer when they log out?**
- **Can the customer directly use the card reader, or does the cashier have to do it?**

Are There Other Formats? A Two-Column Variation

- There isn't one best format; some prefer the one-column style, some the two-column. Sections may be added and removed; heading names may change. Some prefer the two-column or conversational format, which emphasizes the interaction between the actors and the system..

Primary Actor: ...

... as before ...

Main Success Scenario:

Actor Action (or Intention)

1.Customer arrives at a POS checkout with goods and/or services to purchase.

2.Cashier starts a new sale.

3.Cashier enters item identifier.

Cashier repeats steps 3-4 until indicates done.

6.Cashier tells Customer the total, and asks for payment.

7.Customer pays.

System Responsibility

4.Records each sale line item and presents item description and running total.

5.Presents total with taxes calculated.

8.Handles payment.

9.Logs the completed sale and sends information to the external accounting (for all accounting and commissions) and inventory systems (to update inventory). System presents receipt.

GUIDELINES – USEC ASES

- Write Terse Use Cases -Do you like to read lots of requirements? So, write terse use cases.
- Write Black-Box Use Cases -the system is described as having *responsibilities*, which is a common unifying metaphorical theme in object-oriented thinking—software elements have responsibilities and collaborate with other elements that have responsibilities.
- Actor and Actor-Goal Perspective -A set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value *to a particular actor*.

GUIDELINES - USECASES

How to Find Use Cases

- Choose the system boundary. Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?
- Identify the primary actors—those that have goals fulfilled through using services of the system.
- Identify the goals for each primary actor.
- Define use cases that satisfy user goals; name them according to their goal. Usually, user-goal level use cases will be one-to-one with user goals, but there is at least one exception

GUIDELINES - USECASES

Are There Questions to Help Find Actors and Goals?

Who starts and stops the system?

Who does user and security management?

Is there a monitoring process that restarts the system if it fails?

How are software updates handled? Push or pull update?

In addition to *human* primary actors, are there any external software or robotic systems that call upon services of the system?

Who does system administration?

Is "time" an actor because the system does something in response to a time event?

Who evaluates system activity or performance?

Who evaluates logs? Are they remotely retrieved?

Who gets notified when there are errors or failures?

GUIDELINES - USECASES

How to Organize the Actors and Goals?

1. As you discover the results, draw them in a use case diagram, naming the goals as use cases.
2. Write an actor-goal list first, review and refine it, and then draw the use case diagram.

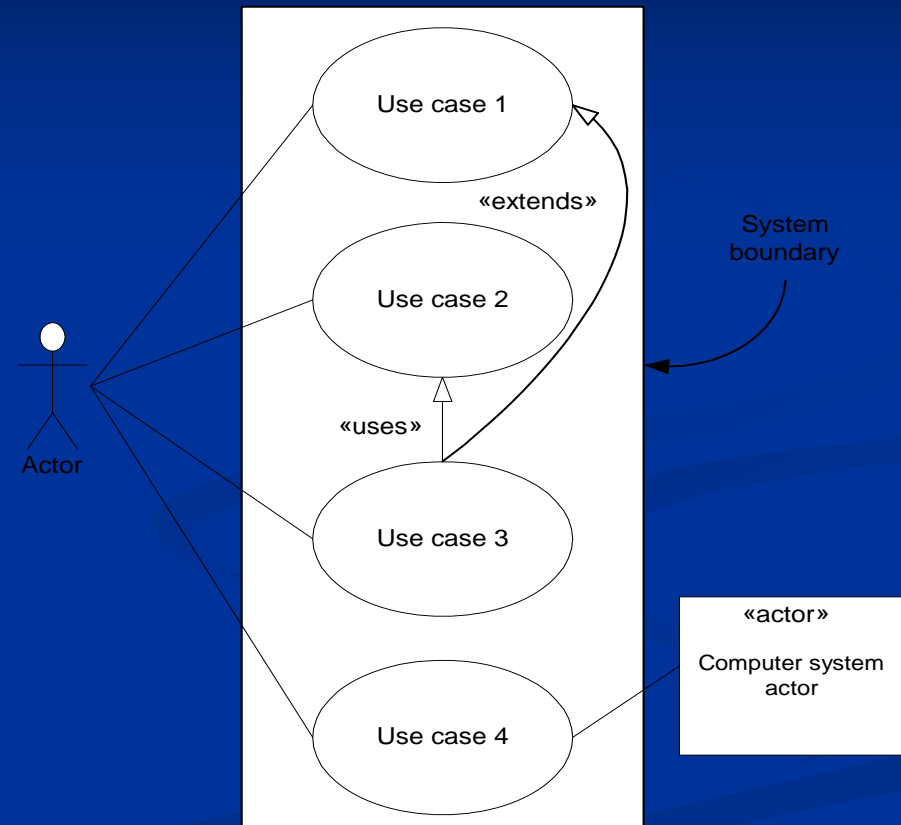
Actor	Goal		Actor	Goal
Cashier	process sales process rentals handle returns cash in cash out ...		System Administrator	add users modify users delete users manage security manage system tables ...
Manager	start up shut down ...		Sales Activity System	analyze sales and performance data
...

Describing the Use-case Model As a Whole

- We need some way to summarize and track the various actors and use cases and their relationships
- Full *use-case model* consists of:
 - Use-case (prose) descriptions, at various appropriate levels of detail
 - Visual model in the form of a use-case diagram
 - Glossary which contains definitions and descriptions of items in the use-case model
 - Select system sequence diagrams

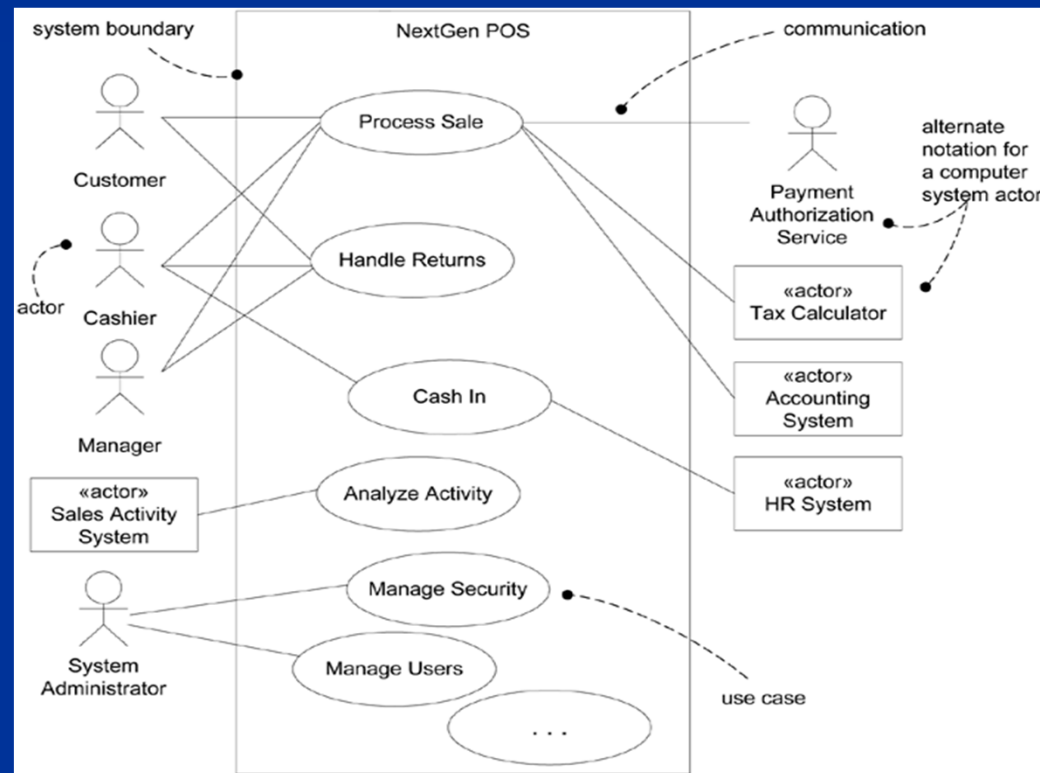
Use-case diagram

- Static, summary illustration of use-case structure.
- Stick-people icons are *actors* that use the system, linked to particular use case(s).
- Rectangles represent *external system* actors that interact with the use case.
- Ellipses indicate use cases themselves.
- «extends» link » specialization
- «uses» link » aggregation
- «include» is same as «uses»



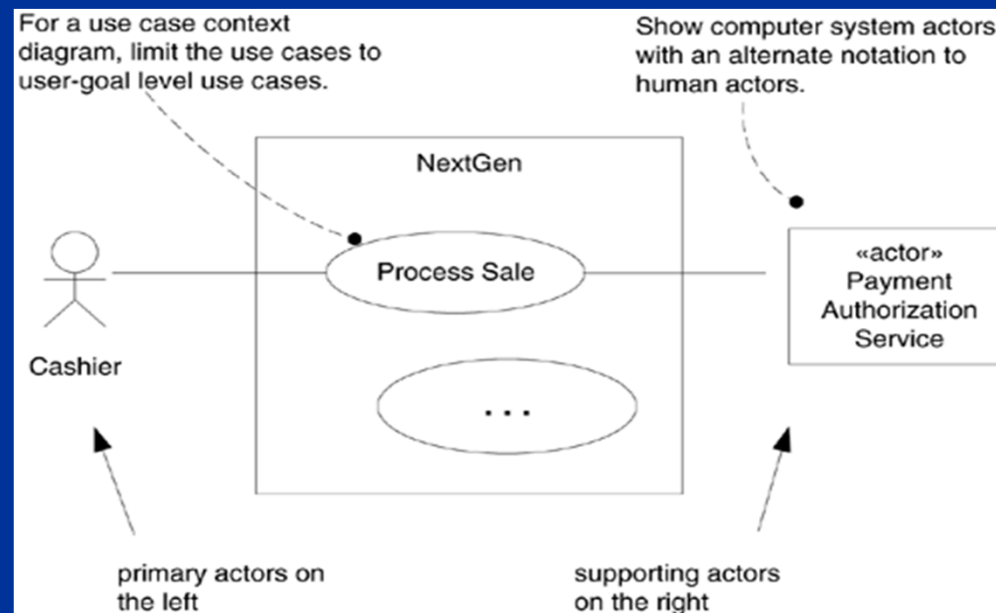
Applying UML: Use Case Diagrams

- The UML provides use case diagram notation to illustrate the names of use cases and actors, and the relationships between them.



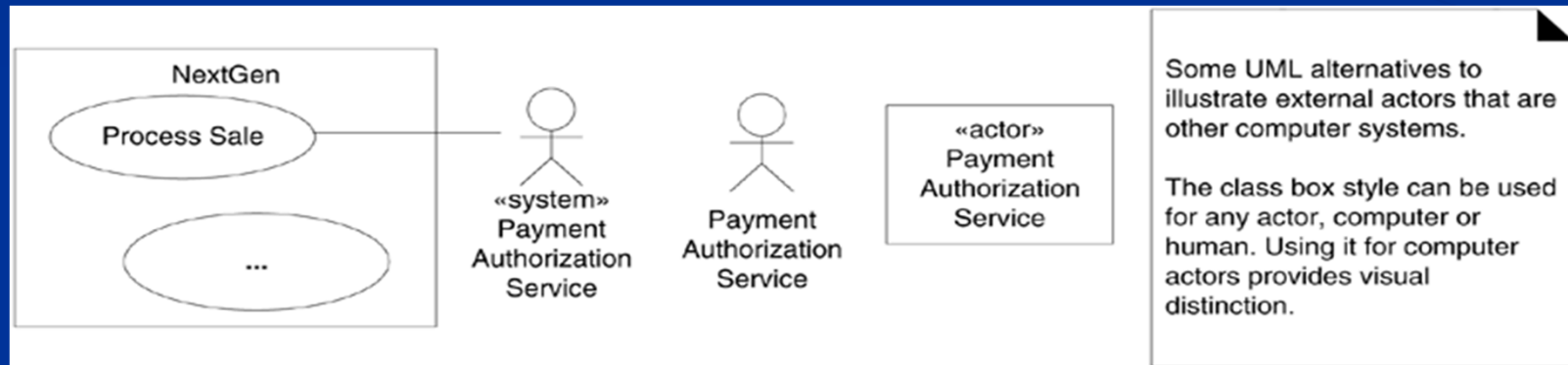
Guideline: Use Case Diagrams

- Notice the actor box with the symbol «actor». This style is used for UML keywords and stereotypes, and includes guillemet symbols—special *single*-character brackets («actor», not <<actor>>) most widely known by their use in



Guideline: Use Case Diagrams

- Some prefer to highlight external computer system actors with an alternate notation-:



High-Level System Feature Lists

- Although detailed function lists are undesirable, a terse, high-level feature list, called system features, added to a Vision document can usefully summarize system functionality.
- Summary of System Features
 - sales capture
 - payment authorization (credit, debit, check)
 - system administration for users, security, code and constants tables, and so on
 - ...

Process: How to Work With Use Cases in Iterative Methods

- Use cases are central to the UP and many other iterative methods.
- The UP encourages use-case driven development. This implies:
 - Functional requirements are primarily recorded in use cases (the Use-Case Model); other requirements techniques (such as functions lists) are secondary, if used at all.
 - Use cases are an important part of iterative planning. The work of an iteration is—in part—defined by choosing some use case scenarios, or entire use cases. And use cases are a key input to estimation.
 - Use-case realizations drive the design. That is, the team designs collaborating objects and subsystems in order to perform or realize the use cases.
 - Use cases often influence the organization of user manuals.
 - Functional or system testing corresponds to the scenarios of use cases.
 - UI "wizards" or shortcuts may be created for the most common scenarios of important use cases to ease common tasks.

How to Evolve Use Cases and Other Specifications Across the Iterations

- Technical team starts building the production core of the system when only perhaps 10% of the requirements are detailed, and in fact, the team deliberately delays in continuing with deep requirements work until near the end of the first elaboration iteration.
- This is a key difference between iterative development and a waterfall process: Production quality development of the core of a system starts quickly, long before all the requirements are known.
- Observe that near the end of the first iteration of elaboration, there is a second requirements workshop, during which perhaps 30% of the use cases are written in detail.
- This staggered requirements analysis benefits from the feedback of having built a little of the core software. The feedback includes user evaluation, testing, and improved "knowing what we don't know." The act of building software rapidly surfaces assumptions and questions that need clarification.

How to Evolve Use Cases and Other Specifications Across the Iterations

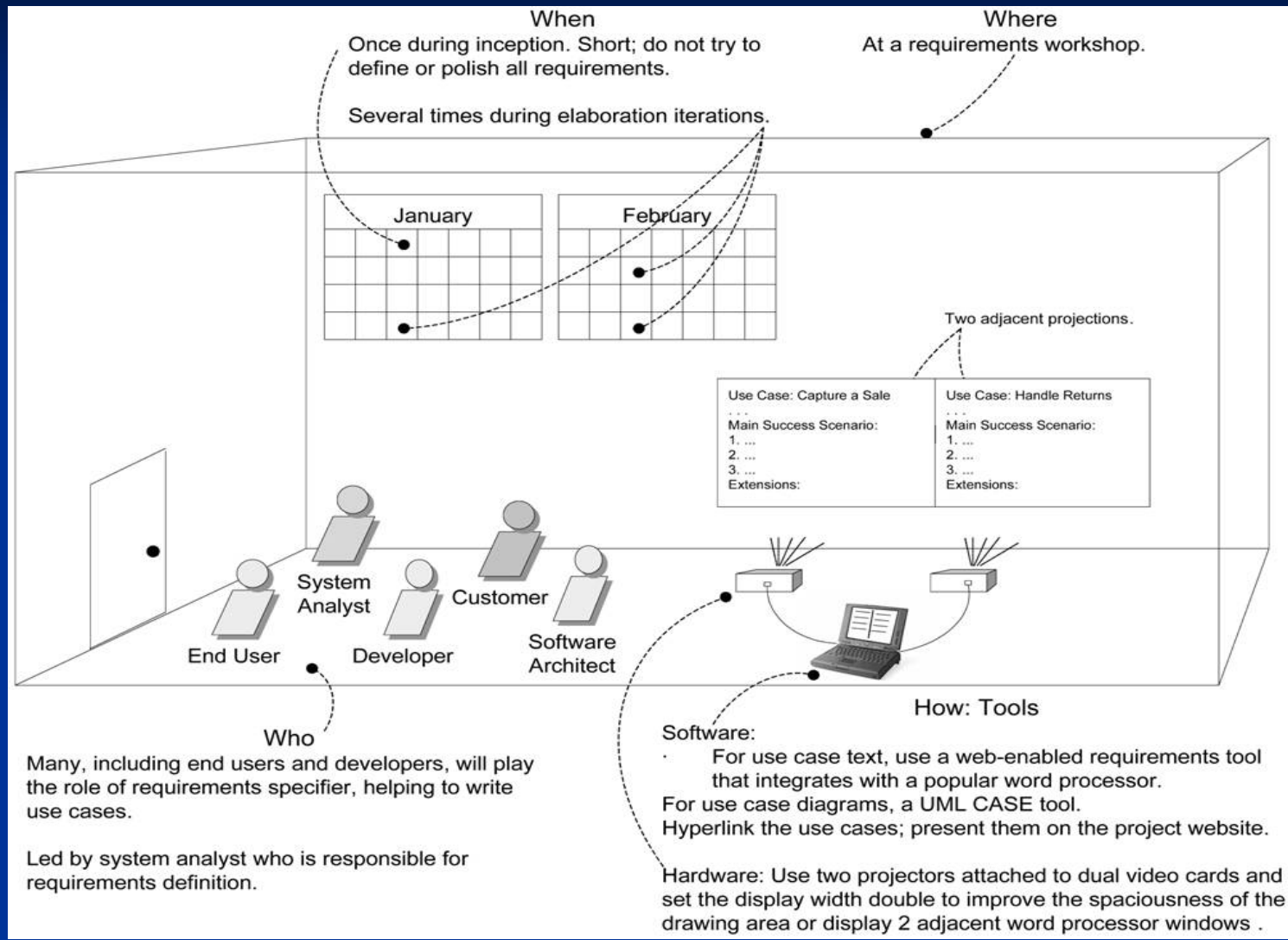
Table below presents a sample (not a recipe) that communicates the UP strategy of how requirements are developed.

Discipline	Artifact	Comments and Level of Requirements Effort				
		Incep 1 week	Elab 1 4 weeks	Elab 2 4 weeks	Elab 3 3 weeks	Elab 4 3 weeks
Requirements	Use-Case Model	2-day requirements workshop. Most use cases identified by name, and summarized in a short paragraph. Pick 10% from the high-level list to analyze and write in detail. This 10% will be the most architecturally important, risky, and high-business value.	Near the end of this iteration, host a 2-day requirements workshop. Obtain insight and feedback from the implementation work, then complete 30% of the use cases in detail.	Near the end of this iteration, host a 2-day requirements workshop. Obtain insight and feedback from the implementation work, then complete 50% of the use cases in detail.	Repeat, complete 70% of all use cases in detail.	Repeat with the goal of 80–90% of the use cases clarified and written in detail. Only a small portion of these have been built in elaboration; the remainder are done in construction.
Design	Design Model	none	Design for a small set of high-risk architecturally significant requirements.	repeat	repeat	Repeat. The high risk and architecturally significant aspects should now be stabilized.
Implementation	Implementation Model (code, etc.)	none	Implement these.	Repeat. 5% of the final system is built.	Repeat. 10% of the final system is built.	Repeat. 15% of the final system is built.
Project Management	SW Development Plan	Very vague estimate of total effort.	Estimate starts to take shape.	a little better...	a little better...	Overall project duration, major milestones, effort, and cost estimates can now be rationally committed to.

How to Evolve Use Cases and Other Specifications Across the Iterations

- Use case writing is encouraged in a requirements workshop.
- Figure below offers suggestions on the time and space for doing this work.

How to Evolve Use Cases and Other Specifications Across the Iterations



When Should Various UP Artifact (Including Use Cases) be Created?

The Use-Case Model is started in inception, with perhaps only 10% of the architecturally significant use cases written in any detail. The majority are incrementally written over the iterations of the elaboration phase, so that by the end of elaboration, a large body of detailed use cases and other requirements (in the Supplementary Specification) are written, providing a realistic basis for estimation through to the end of the project.

Discipline	Artifact	Incep.	Elab.	Const.	Trans.
	Iteration →	I1	E1..En	C1..Cn	T1..T2
Business Modeling	Domain Model		s		
Requirements	Use-Case Model	s	r		
	Vision	s	r		
	Supplementary Specification	s	r		
	Glossary	s	r		
Design	Design Model		s	r	
	SW Architecture Document		s		

Use Cases in RUP

- Lets see the start and refinement of use cases in the following phases in RUP
 1. **Inception**
 2. **Elaboration**
 3. **Construction**

Use Cases in Inception

- Not all use cases are written in their fully dressed format during the inception phase.
- Rather, suppose there is a two-day requirements workshop during the early NextGen investigation. The earlier part of the day is spent identifying goals and stakeholders, and speculating what is in and out of scope of the project.
- An actor-goal-use case table is written and displayed with the computer projector.
- A use case context diagram is started.
- After a few hours, perhaps 20 use cases are identified *by name*.
- Most of the interesting, complex, or risky use cases are written in brief format, each averaging around two minutes to write.
- The team starts to form a high-level picture of the system's functionality.
- After this, 10% to 20% of the use cases that represent core complex functions, require building the core architecture, or that are especially risky in some dimension are rewritten in a fully dressed format;
the team investigates a little deeper to better comprehend the magnitude, complexities, and hidden demons of the project through deep investigation of a small sample of influential use cases.

Use Cases in Elaboration

- This is a phase of multiple timeboxed iterations (for example, four iterations) in which risky, high-value, or architecturally significant parts of the system are incrementally built, and the "majority" of requirements identified and clarified.
- The feedback from the concrete steps of programming influences and informs the team's understanding of the requirements, which are iteratively and adaptively refined.
- Perhaps there is a two-day requirements workshop in each iteration—four workshops. However, not all use cases are investigated in each workshop.
- They are prioritized; early workshops focus on a subset of the most important use cases.
- Each subsequent short workshop is a time to adapt and refine the vision of the core requirements, which will be unstable in early iterations, and stabilizing in later ones.
- Thus, there is an iterative interplay between requirements discovery, and building parts of the software.
- During each requirements workshop, the user goals and use case list are refined.
- More of the use cases are written, and rewritten, in their fully dressed format.
- By the end of elaboration, "80–90%" of the use cases are written in detail.
- For the POS system with 20 user-goal level use cases, 15 or more of the most complex and risky should be investigated, written, and rewritten in a fully dressed format.
- Note that elaboration involves programming parts of the system. At the end of this step, the NextGen team should not only have a better definition of the use cases, but some quality executable software.

Use Cases in Construction

- The construction phase is composed of timeboxed iterations (for example, 20 iterations of two weeks each) that focus on completing the system, once the risky and core unstable issues have settled down in elaboration.
- There may still be some minor use case writing and perhaps requirements workshops, but much less so than in elaboration.

Use Cases in the NextGen Inception Phase

- As described in the previous slides, not all use cases are written in their fully dressed form during inception. The Use-Case Model at this phase of the case study could be detailed as follows:

Fully Dressed	Casual	Brief
Process Sale Handle Returns	Process Rental Analyze Sales Activity Manage Security ...	Cash In Cash Out Manage Users Start Up Shut Down Manage System Tables ...

Case Study : ARENA - The Problem

- The Internet has enabled virtual communities
 - Groups of people sharing common of interests but who have never met each other in person. Such virtual communities can be short lived (e.g people in a chat room or playing a multi player game) or long lived (e.g., subscribers to a mailing list).
- Many multi-player computer games now include support for virtual communities.
 - Players can receive news about game upgrades, new game levels, announce and organize matches, and compare scores.
- Currently each game company develops such community support in each individual game.
 - Each company uses a different infrastructure, different concepts, and provides different levels of support.
- This redundancy and inconsistency leads to problems:
 - High learning curve for players joining a new community,
 - Game companies need to develop the support from scratch
 - Advertisers need to contact each individual community separately.

ARENA: The Objectives

- Provide a generic infrastructure for operating an arena to
 - Support virtual game communities.
 - Register new games
 - Register new players
 - Organize tournaments
 - Keeping track of the players scores.
- Provide a framework for tournament organizers
 - to customize the number and sequence of matchers and the accumulation of expert rating points.
- Provide a framework for game developers
 - for developing new games, or for adapting existing games into the ARENA framework.
- Provide an infrastructure for advertisers.

High-level scenarios identified for ARENA.

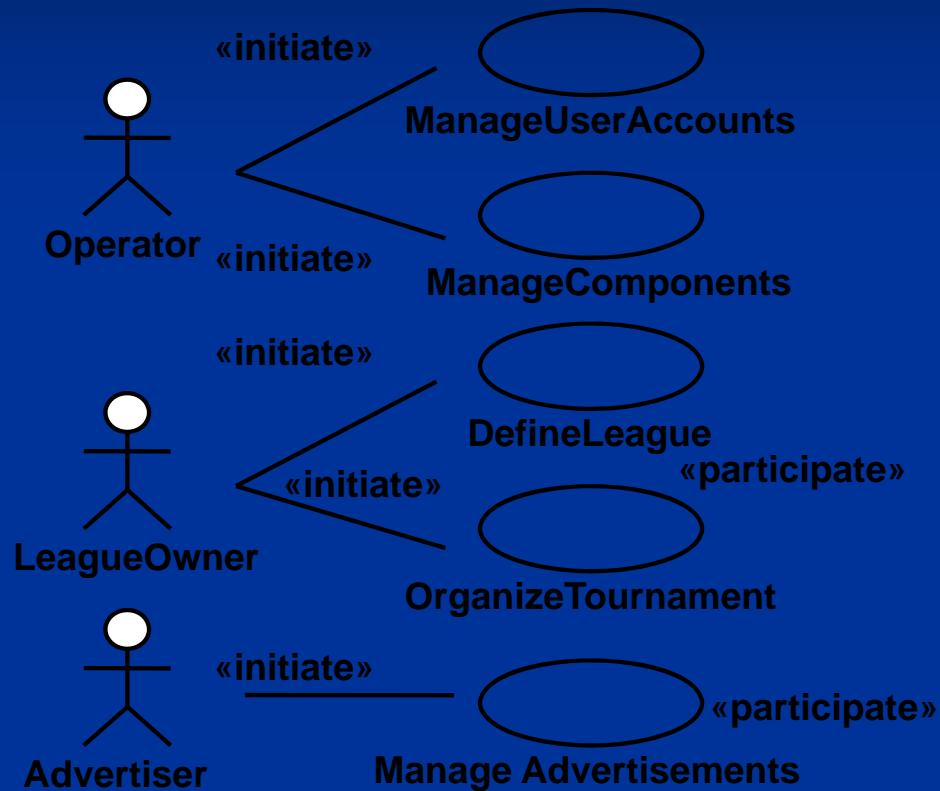


List other actors
List other
scenarios
???

High-level scenarios identified for ARENA.

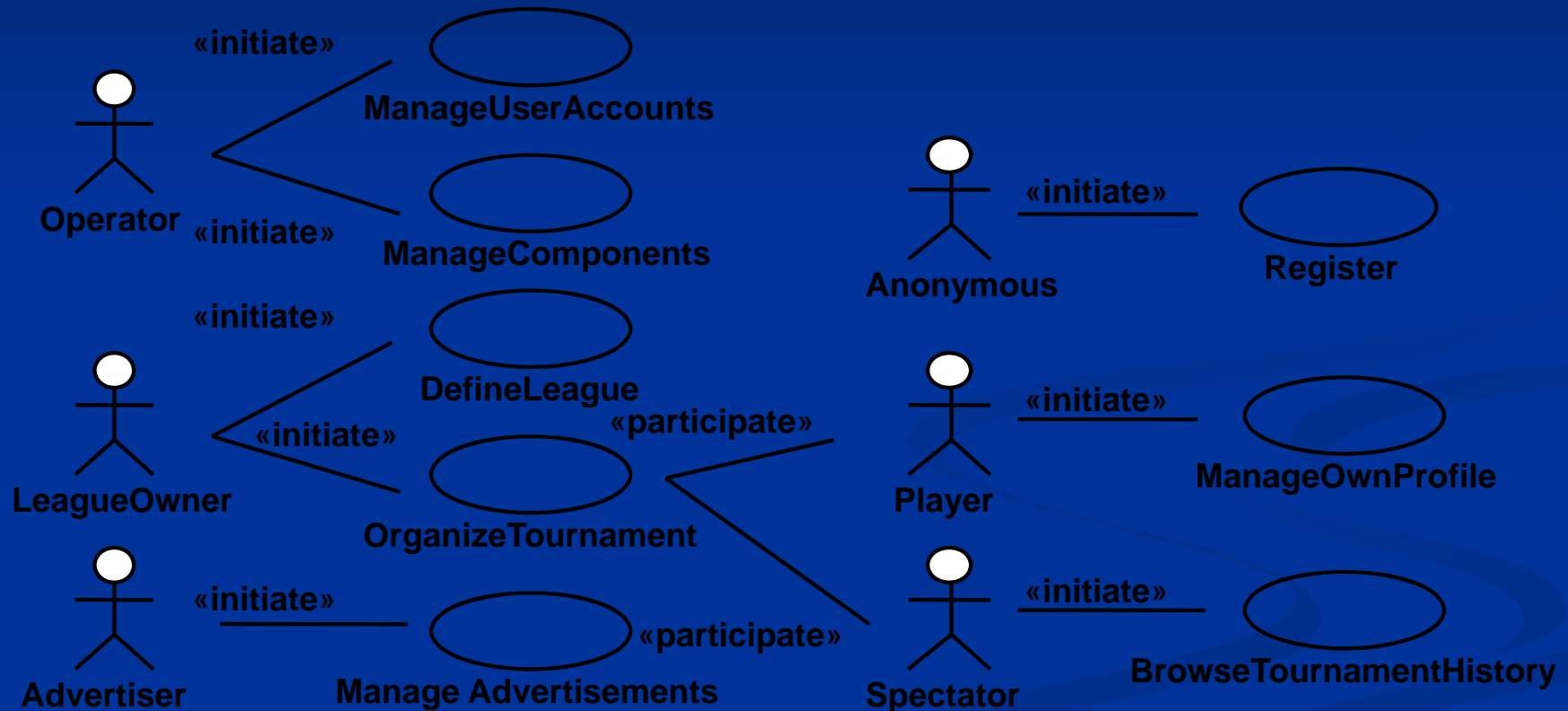


High-level use cases identified for ARENA.

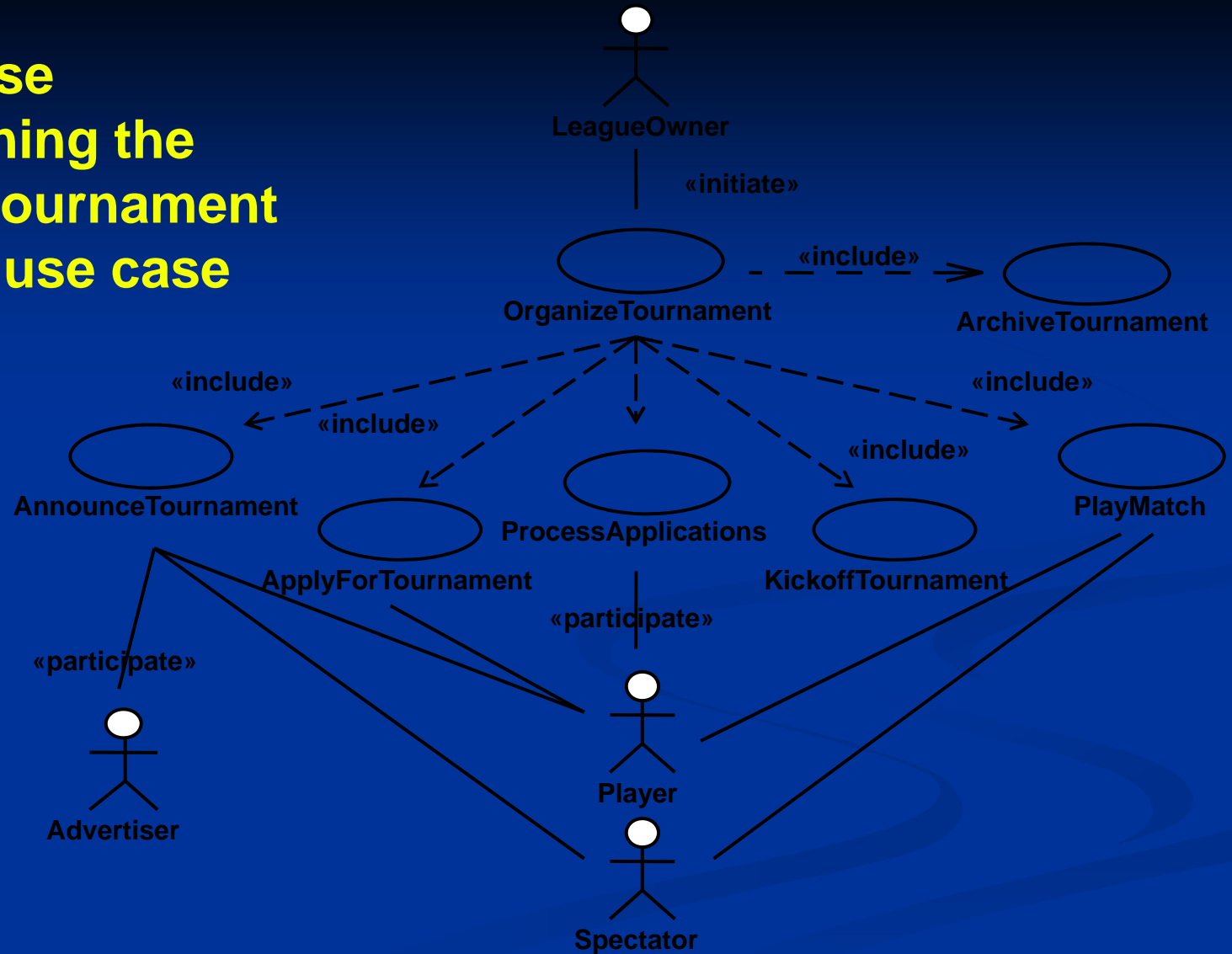


List other actors
List other use
cases
???

High-level use cases identified for ARENA.



Detailed use cases refining the OrganizeTournament high-level use case



System Sequence Diagrams

Introduction

- A system sequence diagram (SSD) is a fast and easily created artifact that illustrates input and output events related to the systems under discussion.
- They are input to operation contracts and most importantly object design.

Introduction

- The use case text and its implied system events are input to SSD creation.
- The SSD operations can in turn be analyzed in the operation contracts, detailed in the Glossary, and most important serve as the starting point for designing collaborating objects.
- An SSD shows, for a particular course of events within a use case, the external actors that interact directly with the system, the system (as a black box), and the system events that the actors generate.

Use Cases and System Sequence Diagrams

- A use case is a prose description of an actor/system interaction.
- Use case diagram is a visual summary of:
 - *All significant use cases.*
 - *All significant actors.*
 - *Relations among actors and use cases.*
- A *System sequence diagram* (SSD) is a visual summary of the individual use cases.
- For ease of understanding, **each use-case scenario corresponds to a separate system sequence diagram.**

Characteristics of System Sequence Diagrams

- System sequence diagrams are a special case of the more-general UML sequence diagram.
- Captures the sequencing of messages and data exchanged between an actor and the system.
- Provides about the same level of detail as use-case prose description.
- Usually deals with primary actors and system, but can include other actors as well.
- Provides a more formal notation for expressing the interaction.
- Concerned *only* with external behavior: the system is treated as a black box.

What is SSD?

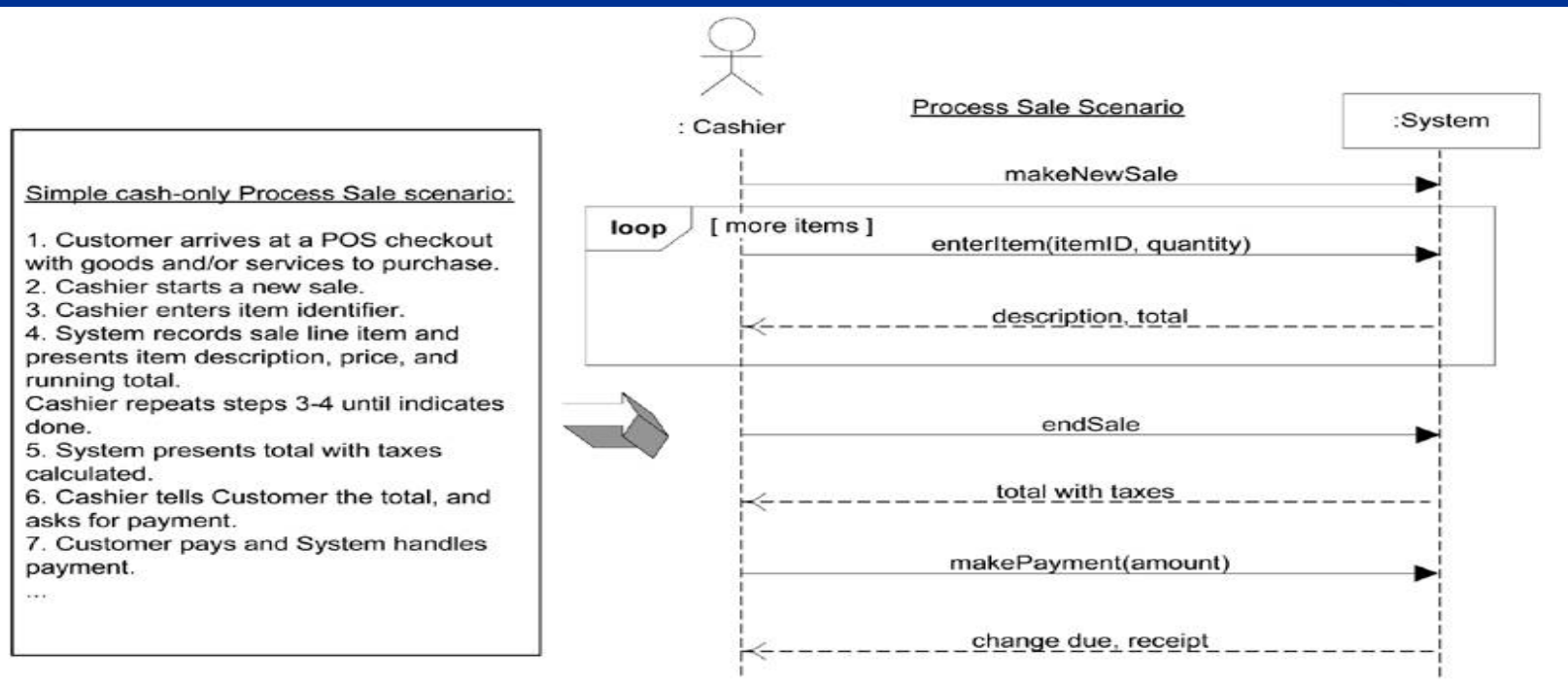
- A system sequence diagram is a picture that shows, for one particular scenario of a use case, the events that external actors generate, their order, and inter-system events.
- All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems.

Why draw an SSD?

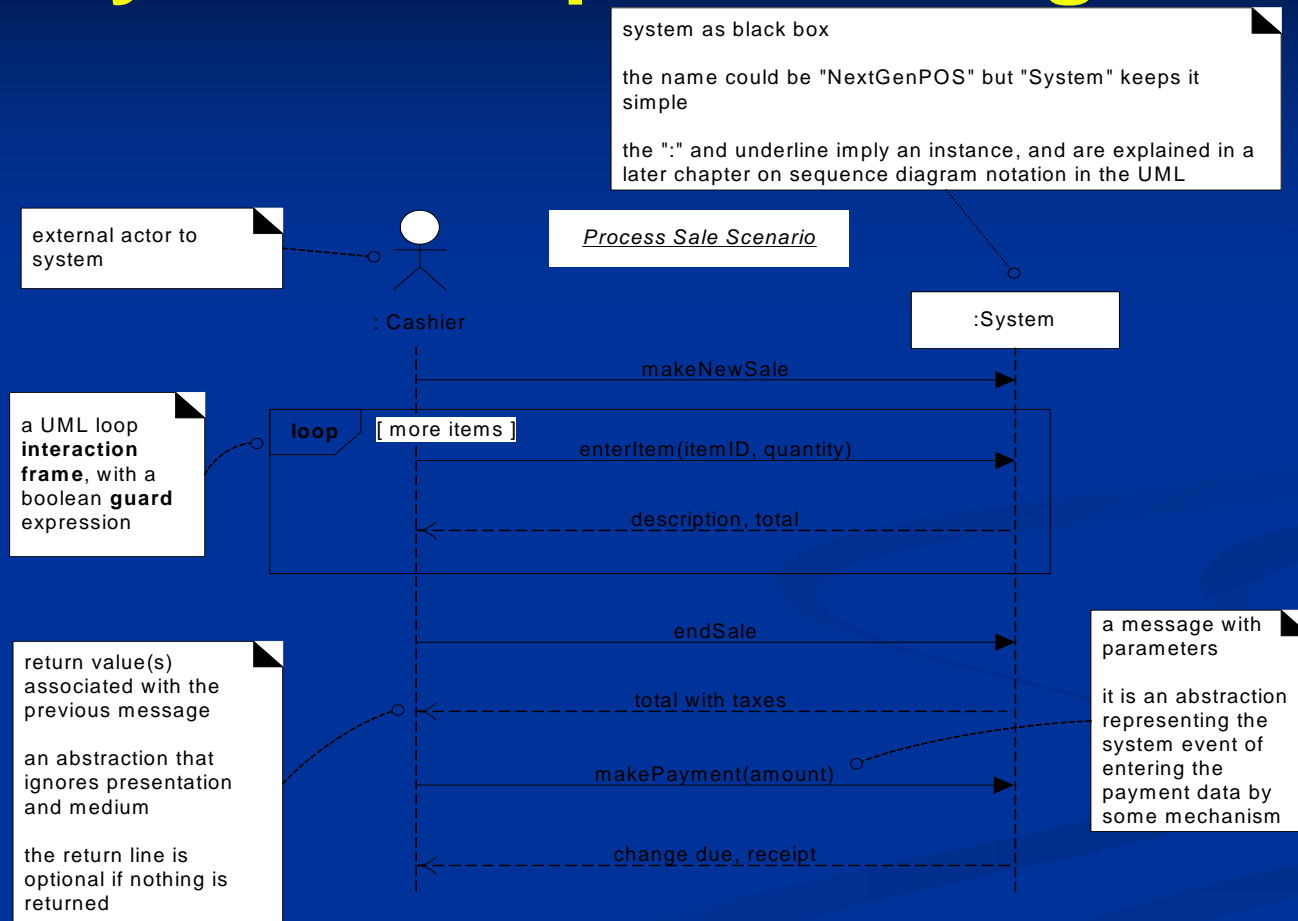
- A software system reacts to three things: 1) external events from actors (humans or computers), 2) timer events, and 3) faults or exceptions (which are often from external sources).
- Therefore, it is useful to know what, precisely, are the external input events the system events. They are an important part of analyzing system behavior.
- System behavior is a description of what a system does, without explaining how it does it. One part of that description is a system sequence diagram.

Relationship Between SSDs and Use Cases?

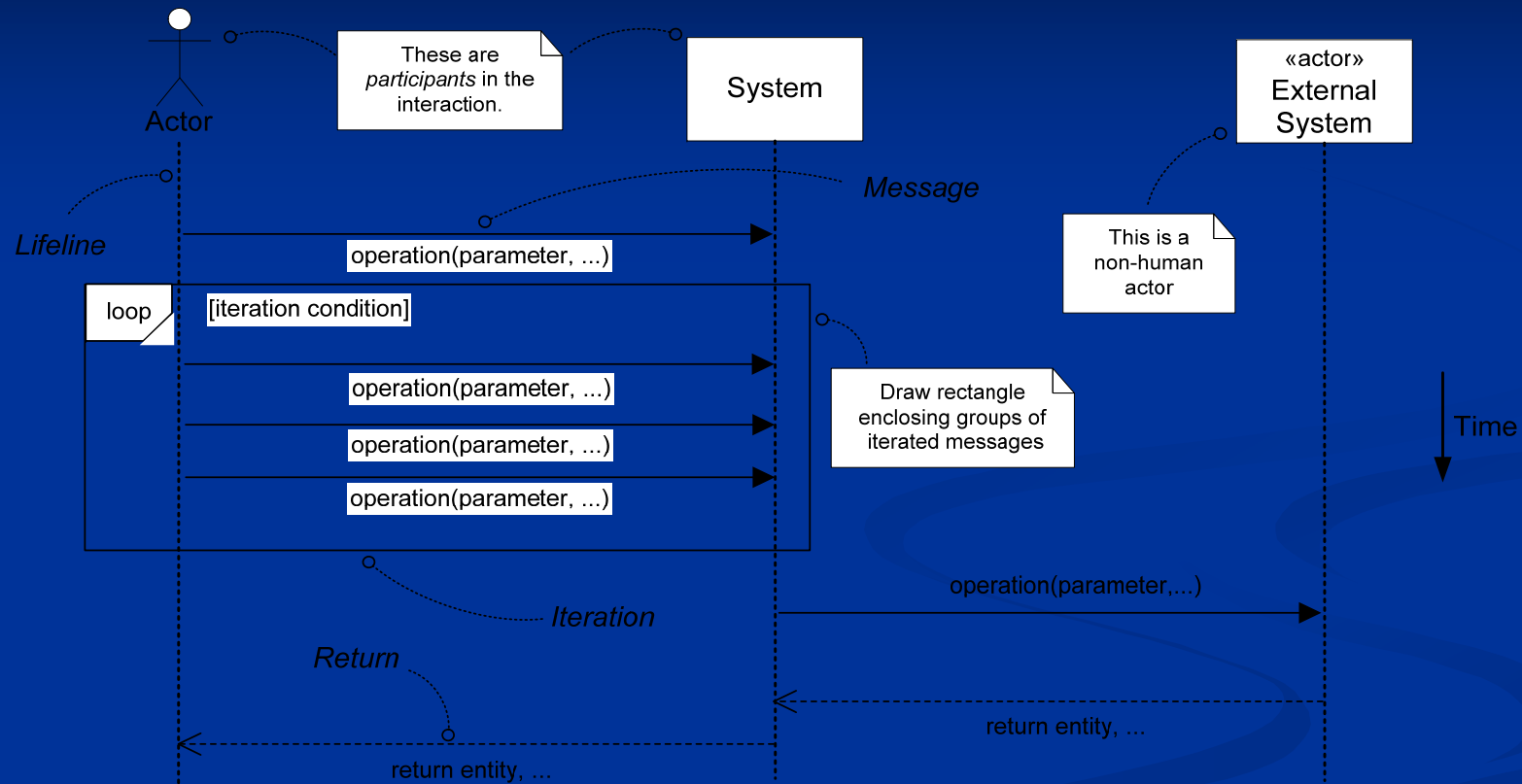
- An SSD shows system events for one scenario of a use case, therefore it is generated from inspection of a use case.



System Sequence Diagram





System Sequence Diagram Notation



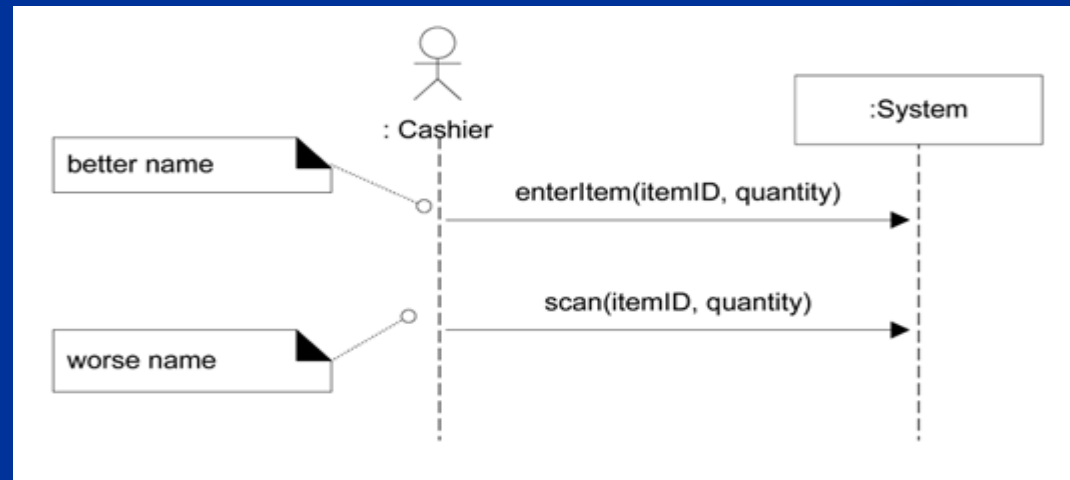
Sequence Diagrams

Messages are indicated by a solid line with an arrow.

Arrow	Meaning	Arrow head style UML 2.0
Synchronous	Transfer control and wait for answer. (sequential operations)	
Return	Returns a value to the caller (optional)	

How to Name System Events and Operations

- System events should be expressed at the abstract level of intention rather than in terms of the physical input device.
- It also improves clarity to start the name of a system event with a verb (add..., enter..., end..., make...), since it emphasizes these are commands or requests.



Process: Iterative and Evolutionary SSDs

■ UP Phases

- **Inception:** SSDs are not usually motivated in inception.
- **Elaboration:** Most SSDs are created during elaboration, when it is useful to identify the details of the system events to clarify what major operations the system must be designed to handle, write system operation contracts, and possibly to support estimation.