Unit 2
Inception

Inception

Defining the vision and obtaining an order-of-magnitude (unreliable) estimate necessitates doing some requirements exploration. However, the purpose of the inception step is not to define all the requirements, or generate a believable estimate or project plan. At the risk of over-simplification, the idea is to do just enough investigation to form a rational, justifiable opinion of the overall purpose and feasibility of the potential new system, and decide if it is worthwhile to invest in deeper exploration (the purpose of the elaboration phase).

Thus, the inception phase should be relatively short for most projects, such as one or a few weeks long. Indeed, on many projects, if it is more than a week long, then the point of inception has been missed: It is to decide if the project is worth a serious investigation (during elaboration), not to do that investigation.

1. **Inception in one sentence:** Envision the product scope, vision, and business case.
2. **The main problem solved in one sentence:** Do the stakeholders have basic agreement on the vision of the project, and is it worth investing in serious investigation?

Artifacts in Inception

Following table lists common inception (or early elaboration) artifacts and indicates the issues they address. A key insight regarding iterative development is to appreciate that these are only partially completed in this phase, will be refined in later iterations, and should not even be created unless it is deemed likely they will add real practical value. And since it is inception, the investigation and artifact content should be light.

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, and related non-functional requirements.
Supplementary Specification	Describes other requirements.
Glossary	Key domain terminology.
Risk List & Risk Management Plan	Describes the business, technical, resource, schedule risks, 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.

For example, the Use-Case Model may list the *names* of most of the expected use cases and actors, but perhaps only describe 10% of the use cases in detail—done in the service of developing a rough high-level vision of the system scope, purpose, and risks. Note that some programming work may occur in inception in order to create "proof of concept" prototypes, to clarify a few requirements via (typically) UI-oriented prototypes, and to do programming experiments for key "show stopper" technical questions.

We didn't understand Inception when:

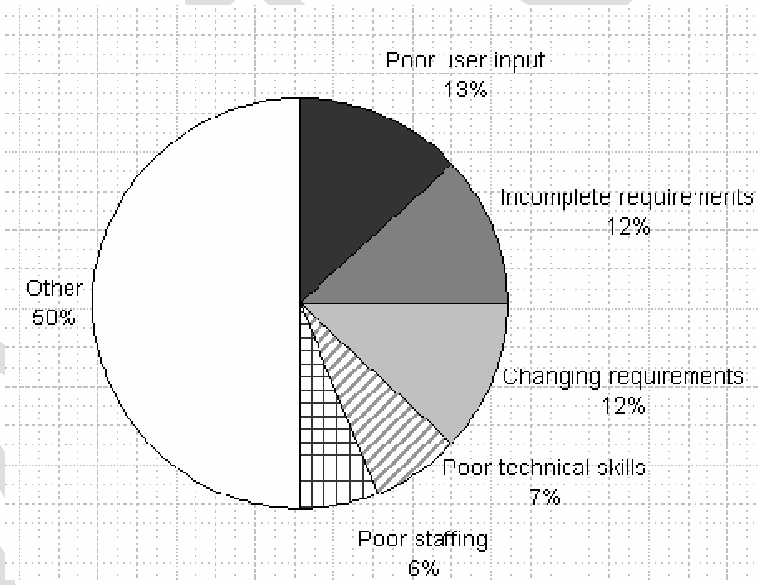
- It is more than "a few" weeks long for most projects.
- There is an attempt to define most of the requirements.

- Estimates or plans are expected to be reliable.
- We define the architecture; rather, this should be done iteratively in elaboration.
- We 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.
- The names of most of the use cases and actors were not identified.
- 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.

Understanding requirements

Requirements are capabilities and conditions to which the system—and more broadly, the project—must conform. A prime challenge of requirements work is to find, communicate, and remember (that usually means record) what is really needed, in a form that clearly speaks to the client and development team members.

The UP promotes a set of best practices, one of which is *manage requirements*. This does not refer to the waterfall attitude of attempting to fully define and stabilize the requirements in the first phase of a project, but rather—in the context of inevitably changing and unclear stakeholder's wishes—"a systematic approach to finding, documenting, organizing, and tracking the changing requirements of a system"; in short, doing it skillfully and not being sloppy. Note the word *changing'*, the UP embraces change in requirements as a fundamental driver on projects. *Finding* is another important term; that is, skillful elicitation via techniques such as use case writing and requirements workshops.



As indicated in the above figure, one study of factors on challenged projects revealed that 37% of factors related to problems with requirements, making requirements issues the largest single contributor to problems. Consequently, masterful requirements management is important. The waterfall response to this data would be to try harder to polish, stabilize, and freeze the requirements before any design or implementation, but history shows this to be a losing battle. The iterative response is to use a process that embraces change and feedback as core drivers in discovering requirements.

FURPS Model / Types of Requirements

In the UP, requirements are categorized according to the FURPS+ model , a useful mnemonic with the following meaning.

1. **Functional**—features, capabilities, security.

2. **Usability**—human factors, help, documentation.
3. **Reliability**—frequency of failure, recoverability, predictability.
4. **Performance**—response times, throughput, accuracy, availability, resource usage.
5. **Supportability**—adaptability, maintainability, internationalization, configurability.

The "+" in FURPS+ indicates ancillary and sub-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**
- **Legal**—licensing and so forth.

It is helpful to use FURPS+ categories (or some categorization scheme) as a checklist for requirements coverage, to reduce the risk of not considering some important facet of the system. Some of these requirements are collectively called the **quality attributes, quality requirements,** or the "-ilities" of a system. These include usability, reliability, performance, and supportability. In common usage, requirements are categorized as **functional** (behavioral) or **non-functional** (everything else); some dislike this broad generalization, but it is very widely used.

Functional requirements are explored and recorded in the Use-Case Model and in the system features list of the Vision artifact. Other requirements can be recorded in the use cases they relate to, or in the Supplementary Specifications artifact. The Vision artifact summarizes high-level requirements that are elaborated in these other documents. The Glossary records and clarifies terms used in the requirements. The Glossary in the UP also encompasses the concept of the **data dictionary,** which records requirements related to data, such as validation rules, acceptable values, and so forth. Prototypes are a mechanism to clarify what is wanted or possible.

As we shall see when exploring architectural analysis, the quality requirements have a strong influence on the architecture of a system. For example, a high-performance, high-reliability requirement will influence the choice of software and hardware components, and their configuration. The need for easy adaptability due to frequent changes in the functional requirements would likewise fundamentally shape the design of the software.

Understanding the Use case Model

The UP defines the Use-Case Model within the Requirements discipline. Essentially, this is the set of all use cases; it is a model of the system's functionality and environment. The idea of use cases to describe functional requirements was introduced in 1986 by Ivar Jacobson, a main contributor to the UML and UP. Jacobson's use case idea was seminal and widely appreciated; simplicity and utility being its chief virtues.

Use Cases and Adding Value

A scenario is a specific sequence of actions and interactions between actors and the system under discussion; it is also called a **use case instance.** It is one particular story of using a system, or one path through the use case; for example, the scenario of successfully purchasing items with cash, or the scenario of failing to purchase items because of a credit card transaction denial. Informally then, a **use case** is a collection of related success and failure scenarios that describe actors using a system to support a goal. For example, here is a *casual format* use case that includes some alternate scenarios:

Main Success Scenario:

- A customer arrives at a checkout with items to return. The cashier uses the POS system to record each returned item...

Alternate Scenarios:

- If the credit authorization is reject, inform the customer and ask for an alternate payment method.
- If the item identifier is not found in the system, notify the Cashier and suggest manual entry of the identifier code (perhaps it is corrupted).

- If the system detects failure to communicate with the external tax calculator system,...

Use Cases and Functional Requirements

Use cases are requirements; primarily they are functional requirements that indicate what the system will do. In terms of the FURPS+ requirements types, they emphasize the "F" (functional or behavioural), but can also be used for other types, especially when those other types strongly relate to a use case. In the UP—and most modern methods—use cases are the central mechanism that is recommended for their discovery and definition. Use cases define a promise or contract of how a system will behave.

Use cases *are* requirements (although not all requirements). Some think of requirements only as "the system shall do..." function or feature lists. Not so, and a key idea of use cases is to (usually) reduce the importance or use of detailed older-style feature lists and rather, write use cases for the functional requirements.

Use cases are text documents, not diagrams, and use-case modelling is primarily an act of writing text, not drawing. However, the UML defines a use case diagram to illustrate the names of use cases and actors, and their relationships.

Use Case Types and Formats

Black-Box Use Cases and System Responsibilities

Black-box use cases are the most common and recommended kind; they do not describe the internal workings of the system, its components, or design. Rather, 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.

By defining system responsibilities with black-box use cases, it is possible to specify *what* the system must do (the functional requirements) without deciding *how* it will do it (the design). Indeed, the definition of "analysis" versus "design" is sometimes summarized as "what" versus "how." This is an important theme in good software development: During requirements analysis avoid making "how" decisions, and specify the external behaviour for the system, as a black box. Later, during design, create a solution that meets the specification.

Black-box style	**Not**
The system records the sale.	The system writes the sale to a database. or (even worse): The system generates a SQL INSERT statement for the sale...

Formality Types:

Use cases are written in different formats, depending on need. In addition to the black-box versus white-box *visibility* type, use cases are written in varying degrees *of formality:*

- **Brief**—terse one-paragraph summary, usually of the main success scenario. The prior *Process Sale* example was brief.
- **Casual**—informal paragraph format. Multiple paragraphs that cover various scenarios. The prior *Handle Returns* example was casual.
- **Fully dressed**—the most elaborate. All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.

Writing use cases - Goals and Scope of a Use Case

How should use cases be discovered? It is common to be unsure if something is a valid (or more practically, a useful) use case. Tasks can be grouped at many levels of granularity, from one or a few small steps, up to enterprise-level activities.

At what level and scope should use cases be expressed?

The following sections examine the simple ideas of elementary business processes and goals as a framework for identifying the use cases for an application.

Use Cases for Elementary Business Processes

Which of these is a valid use case?

- Negotiate a Supplier Contract
- Handle Returns
- Log In

An argument can be made that all of these are use cases *at different levels*, depending on the system boundary, actors, and goals. Evaluation of these candidates is presented after an introduction to elementary business processes. Rather than asking in general, "What is a valid use case?", question for the POS case study is: What is a useful level to express use cases for application requirements analysis?

The EBP Use Case: For requirements analysis for a computer application, focus on use cases at the level **of elementary business processes** (EBPs).

EBP is a term from the business process engineering field,4 defined as:

A task performed by one person in one place at one time, in response to a business event, which adds measurable business value and leaves the data in a consistent state. e.g., Approve Credit or Price Order. This can be taken too literally: Does a use case fail as an EBP if two people are required, or if a person has to walk around? Probably not, but the feel of the definition is about right. It's not a single small step like "delete a line item" or "print the document." Rather, the main success scenario is probably five or ten steps. It doesn't take days and multiple sessions, like "negotiate a supplier contract;" it is a task done during a single session. It is probably between a few minutes and an hour in length. As with the UP's definition, it emphasizes adding observable or measurable business value, and it comes to a resolution in which the system and data are in a stable and consistent state.

A common use case mistake is defining many use cases at too low a level; that is, as a single step, sub-function, or subtask within an EBP.

Reasonable Violations of the EBP Guideline

Although the "base" use cases for an application should satisfy the EBP guideline, it is frequently useful to create separate "sub" use cases representing sub-tasks or steps within a base use case. Use cases can exist that fail the EBP test; many potentially exist at a lower level. The guideline is only used to find the dominant level of use cases in requirements analysis for an application; that is, the level to focus on for naming and writing them.

For example, a subtask or extension such as "paying by credit" may be repeated in several base use cases. It is desirable to separate this into its own use case (that does not satisfy the EBP guideline) and link it to several base use cases, to avoid duplication of the text.

Use Cases and Goals

Actors have goals (or needs) and use applications to help satisfy them. Consequently, an EBP-level use case is called a **user** goal-level user case, to emphasize that it serves (or should serve) to fulfill a goal of a user of the system, or the primary actor.

And it leads to a recommended procedure:

1. Find the user goals.
2. Define a use case for each.

This is slight shift in emphasis for the use-case modeler. Rather than asking "What are the use cases?", one starts by asking: "What are your goals?" In fact, the name of a use case for a user goal should reflect its name, to emphasize this view point.

Goal: capture or process a sale; use case: *Process Sale*

Note that because of this symmetry, the EBP guideline can be equally applied to decide if a goal or a use case is at a suitable level. Thus, here is a key idea regarding investigating user goals vs. investigating use cases:

Imagine we are together in a requirements workshop. We could ask either:

- "What do you do?" (roughly a use case-oriented question) or,
- "What are your goals?"

Answers to the first question are more likely to reflect current solutions and procedures, and the complications associated with them.

Answers to the second question, especially combined with an investigation to move higher up the goal hierarchy ("what is the goal of that goal?") open up the vision for new and improved solutions, focus on adding business value, and get to the heart of what the stakeholders want from the system under discussion.

Example: Applying the EBP Guideline

As the system analyst responsible for the NextGen system requirements discovery, you are investigating user goals. The conversation goes like this: During a requirements workshop:

System analyst: "What are some of your goals in the context of using a POS system?"

Cashier: "One, to quickly log in. Also, to capture sales."

System analyst: "What do you think is the higher level goal motivating logging in?"

Cashier: "I'm trying to identify myself to the system, so it can validate that I'm allowed to use the system for sales capture and other tasks."

System analyst: "Higher than that?"

Cashier: "To prevent theft, data corruption, and display of private company information."

Note the analyst's strategy of searching up the goal hierarchy to find higher level user goals that still satisfy the EBP guideline, to get at the real intent behind the action, and also to understand the context of the goals. "Prevent theft, ..." is higher than a user goal; it may be called an enterprise goal, and is not an EBP. Therefore, although it can inspire new ways of thinking about the problem and solutions (such as eliminating POS systems and cashiers completely), we will set it aside for now.

Lowering the goal level to "identify myself and be validated" appears closer to the user goal level. But is it at the EBP level? It does not add observable or measurable business value. If the CEO asked, "What did you do today?" and you said "I logged in 20 times!", she would not be impressed. Consequently, this is a secondary goal, always in the service of doing something useful, and is not an EBP or user goal. By contrast, "capture a sale" does fit the criteria of being an EBP or user goal.

As another example, in some stores there is a process called "cashing in", in which a cashier inserts their own cash drawer tray into the terminal, logs in, and tells the system how much cash is in drawer. *Cashing In* is an EBP-level (or user goal level) use case; the log in step, rather than being a EBP-level use case, is a subfunction goal in support of the goal of cashing in.

Sub-function Goals and Use Cases

Although "identify myself and be validated" (or "log in") has been eliminated as a user goal, it is a goal at a lower level, called a **sub-function goal**. Sub-goals that support a user goal. Use cases should only occasionally be written for these sub-function goals, although it is a common problem that use case experts observe when asked to evaluate and improve (usually simplify) a set of use cases.

It is not illegal to write use cases for sub-function goals, but it is not always helpful, as it adds complexity to a use-case model; there can be hundreds of sub-function goals or sub-function use cases for a system. Important point: The number and granularity of use cases influences the time and difficulty to understand, maintain, and manage the requirements. The most common, valid motivation to express a sub-function goal as a use case is when the sub-function is repeated in or is a precondition for multiple user goal-level use cases. This in fact is probably true of "identify myself and be validated," which is a precondition of most, if not all, other user goal-level use cases. Consequently, it may be written as the use case *Authenticate User.*

Goals and Use Cases Can Be Composite

Goals are usually composite, from the level of an enterprise ("be profitable"), to many supporting intermediate goals while using applications ("sales are captured"), to supporting sub-function goals within applications ("input is valid").Similarly, use cases can be written at different levels to satisfy these goals, and can be composed of lower level use cases.

These varying goal and use case levels are a common source of confusion in identifying the appropriate level of use cases for an application. The EBP guideline provides guidance to filter out excessive low-level use cases.

Elements / sections of a use case

Preface Elements

Many optional preface elements are possible. Only place elements at the start which are important to read before the main success scenario. Move extraneous "header" material to the end of the use case.

Primary Actor: The principal actor that calls upon system services to fulfill a goal.

Important: Stakeholders and Interests List

This list is more important and practical than may appear at first glance. It suggests and bounds what the system must do. To quote:

The [system] operates a contract between stakeholders, with the use cases detailing the behavioral parts of that contract...The use case, as the contract for behavior, captures *all and only* the behaviors related to satisfying the stakeholders' interests.

This answers the question: What should be in the use case? The answer is: That which satisfies all the stakeholders' interests. In addition, by starting with the stakeholders and their interests before writing the remainder of the use case, we have a method to remind us what the more detailed responsibilities of the system should be. For example, would I have identified a responsibility for salesperson commission handling if I had not first listed the salesperson stakeholder and their interests? Hopefully eventually, but perhaps I would have missed it during the first analysis session. The stakeholder interest viewpoint provides a thorough and methodical procedure for discovering and recording all the required behaviors.

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.

Preconditions and Success Guarantees (Postconditions)

Preconditions state what *must always* be true before beginning a scenario 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 that has successfully completed, such as logging in, or the more general "cashier is identified and authenticated." Note that there are conditions that must be true, but are not of practical value to write, such as "the system has power." Preconditions communicate noteworthy assumptions that the use case writer thinks readers should be alerted to.

Success guarantees (or **postconditions)** 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.

Preconditions: Cashier is identified and authenticated.

Success Guarantee (Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated.

Main Success Scenario and Steps (or Basic Flow)

This has also been called the "happy path" scenario, or the more prosaic "Basic Flow." It describes the typical success path that satisfies the interests of the stakeholders. Note that 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.

Suggestion

Defer all conditional and branching statements to the Extensions section. The scenario records the steps, of which there are three kinds:

1. An interaction between actors.3
2. A validation (usually by the system).
3. A state change by the system (for example, recording or modifying something).

Step one of a use case does not always fall into this classification, but indicates the trigger event that starts the scenario. It is a common idiom to always capitalize the actors' names for ease of identification. Observe also the idiom that is used to indicate repetition.

Main Success Scenario:

1. Customer arrives at a POS checkout with items to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. Cashier repeats steps 3-4 until indicates done.
5. ...

Extensions (or Alternate Flows)

Extensions are very important. They indicate all the other scenarios or branches, both success and failure. Observe in the fully dressed example that the Extensions section was considerably longer and more complex than the Main Success Scenario section; this is common and to be expected. They are also known as "Alternative Flows."

In thorough use case writing, the combination of the happy path and extension scenarios should satisfy "nearly" all the interests of the stakeholders. This point is qualified, because some interests may best be captured as non-functional requirements expressed in the Supplementary Specification rather than the use cases.

Extension scenarios are branches from the main success scenario, and so can be notated with respect to it. For example, at Step 3 of the main success scenario there may be an invalid item identifier, either because it was incorrectly entered or unknown to the system. An extension is labeled "3a"; it first identifies the condition and then the response. Alternate extensions at Step 3 are labeled "3b" and so forth.

Extensions:

3a. Invalid identifier:

1. System signals error and rejects entry.

3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers): 1. Cashier can enter item category identifier and the quantity.

An extension has two parts: the condition and the handling.

Guideline: Write the condition as something that can be *detected* by the systemor an actor. To contrast:

5a. System detects failure to communicate with external tax calculation system service:

5a. External tax calculation system not working:

The former style is preferred because this is something the system can detect; the latter is an inference.

Extension handling can be *summarized* in one step, or include a sequence, as in this example, which also illustrates notation to indicate that a condition can arise within a range of steps:

3-6a: Customer asks Cashier to remove an item from the purchase:

1. Cashier enters the item identifier for removal from the sale.
2. System displays updated running total.

At the end of extension handling, by default the scenario merges back with the main success scenario, unless the extension indicates otherwise (such as by halting the system).

Sometimes, a particular extension point is quite complex, as in the "paying by credit" extension. This can be a motivation to express the extension as a separate use case.

This extension example also demonstrates the notation to express failures within extensions.

7b. Paying by credit:

1. Customer enters their credit account information.
2. System requests payment validation from external Payment Authorization Service System.

2a. System detects failure to collaborate with external system:

1. System signals error to Cashier.
2. Cashier asks Customer for alternate payment.
3. ...

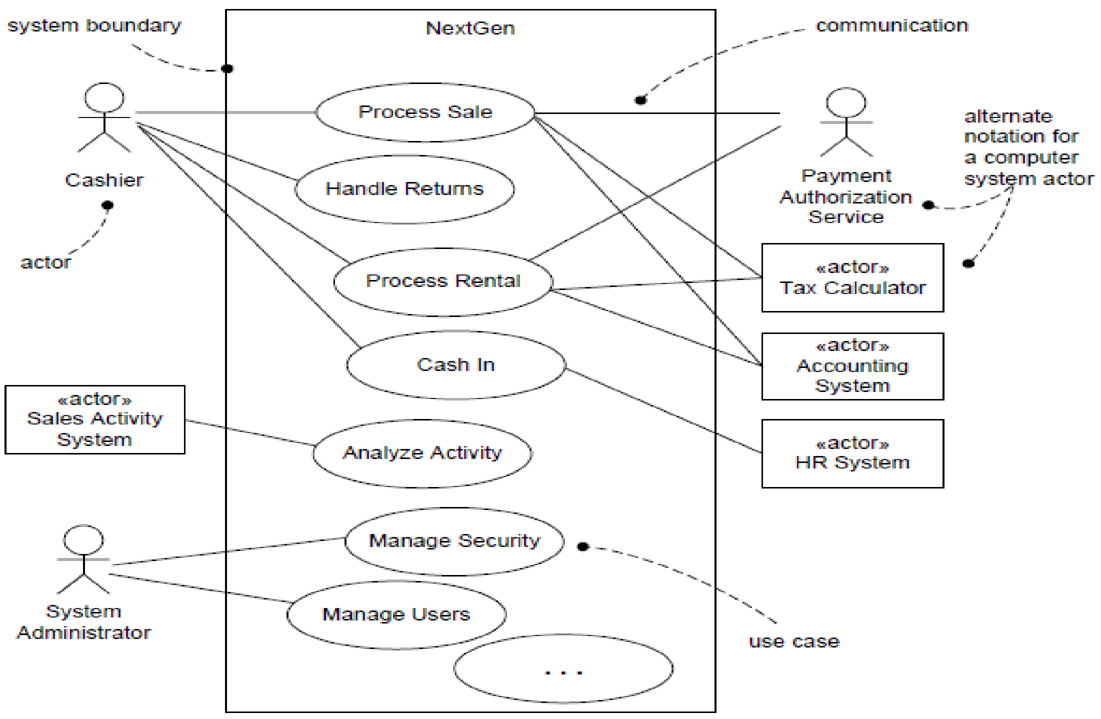
If it is desirable to describe an extension condition as possible during any (or at least most) steps, the labels *a, *b, ..., can be used. *a. At any time, System crashes:
In order to support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered at any step in the scenario.
1. Cashier restarts the System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.
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.
Special Requirements:

- Touch screen Ul on a large flat panel monitor. Text must be visible from 1 meter.
- Credit authorization response within 30 seconds 90% of the time.
- Language internationalization on the text displayed.
- Pluggable business rules to be insertable at steps 2 and 6.

Recording these with the use case is classic UP advice, and a reasonable location when first writing the use case. However, many practitioners find it useful to ultimately consolidate all non-functional requirements in the Supplementary Specification, for content management, comprehension, and readability, because these requirements usually have to be considered as a whole during architectural analysis.
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.
For example, a stakeholder might say, "The POS system must support credit account input using a card reader and the keyboard." Note that these are examples of early design decisions or constraints; in general, it is skillful to avoid premature design decisions, but sometimes they are obvious or unavoidable, especially concerning input/output technologies.
It is also necessary to understand variations in data schemes, such as using UPCs or EANs for item identifiers, encoded in bar code symbology.
This list is the place to record such variations. It is also useful to record variations in the data that may be captured at a particular step.
Technology and Data Variations List:
3a. Item identifier entered by laser scanner 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.
Suggestion
This section should *not* contain multiple steps to express varying behavior for different cases. If that is necessary, say it in the Extensions section.

Use case Diagram

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



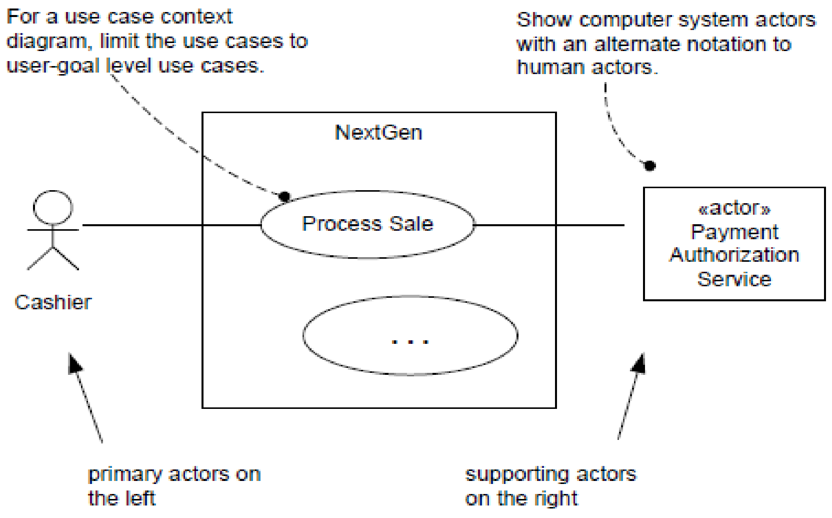
Use case diagrams and use case relationships are secondary in use case work. Use cases are text documents. Doing use case work means to write text. A common sign of a novice (or academic) use-case modeler is a preoccupation with use case diagrams and use case relationships, rather than writing text. World-class use case experts such as Anderson, Fowler, Cockburn, among others, downplay use case diagrams and use case relationships, and instead focus on writing. With that as a caveat, a simple use case diagram provides a succinct visual context diagram for the system, illustrating the external actors and how they use the system.

Suggestion

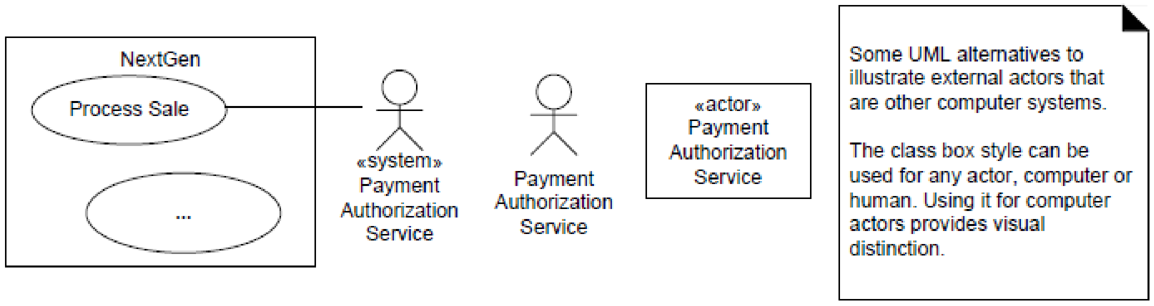
Draw a simple use case diagram in conjunction with an actor-goal list. A use case diagram is an excellent picture of the system context; it makes a good **context diagram that** is, showing the boundary of a system, what lies outside of it, and how it gets used. It serves as a communication tool that summarizes the behavior of a system and its actors. A sample *partial* use case context diagram for the NextGen system is shown in Figure.

Diagramming Suggestions

Figure offers some diagram advice. Notice the actor box with the symbol «actor». This symbol is called a UML **stereotype;** it is a mechanism to categorize an element in some way. A stereotype name is surrounded by guillemets symbols.special single-character brackets (not "«" and "»") most widely known by their use in French typography to indicate a quote.


For a use case context diagram, limit the use cases to user-goal level use cases.
Show computer system actors with an alternate notation to human actors.
NextGen
Process Sale
. . .
Cashier
«actor» Payment Authorization Service
primary actors on the left
supporting actors on the right

To clarify, some prefer to highlight external computer system actors with an alternate notation, as illustrated in Figure.


NextGen
Process Sale
...
«system» Payment Authorization Service
Payment Authorization Service
«actor» Payment Authorization Service
Some UML alternatives to illustrate external actors that are other computer systems.
The class box style can be used for any actor, computer or human. Using it for computer actors provides visual distinction.

A Caution on Over-Diagramming

To reiterate, the important use case work is to write text, not diagram or focus on use case relationships. If an organization is spending many hours (or worse, days) working on a use case diagram and discussing use case relationships, rather than focussing on writing text, relative effort has been misplaced.

Use Cases With in the UP

Use cases are vital and central to the UP, which encourages **use-case driven development.** This implies:

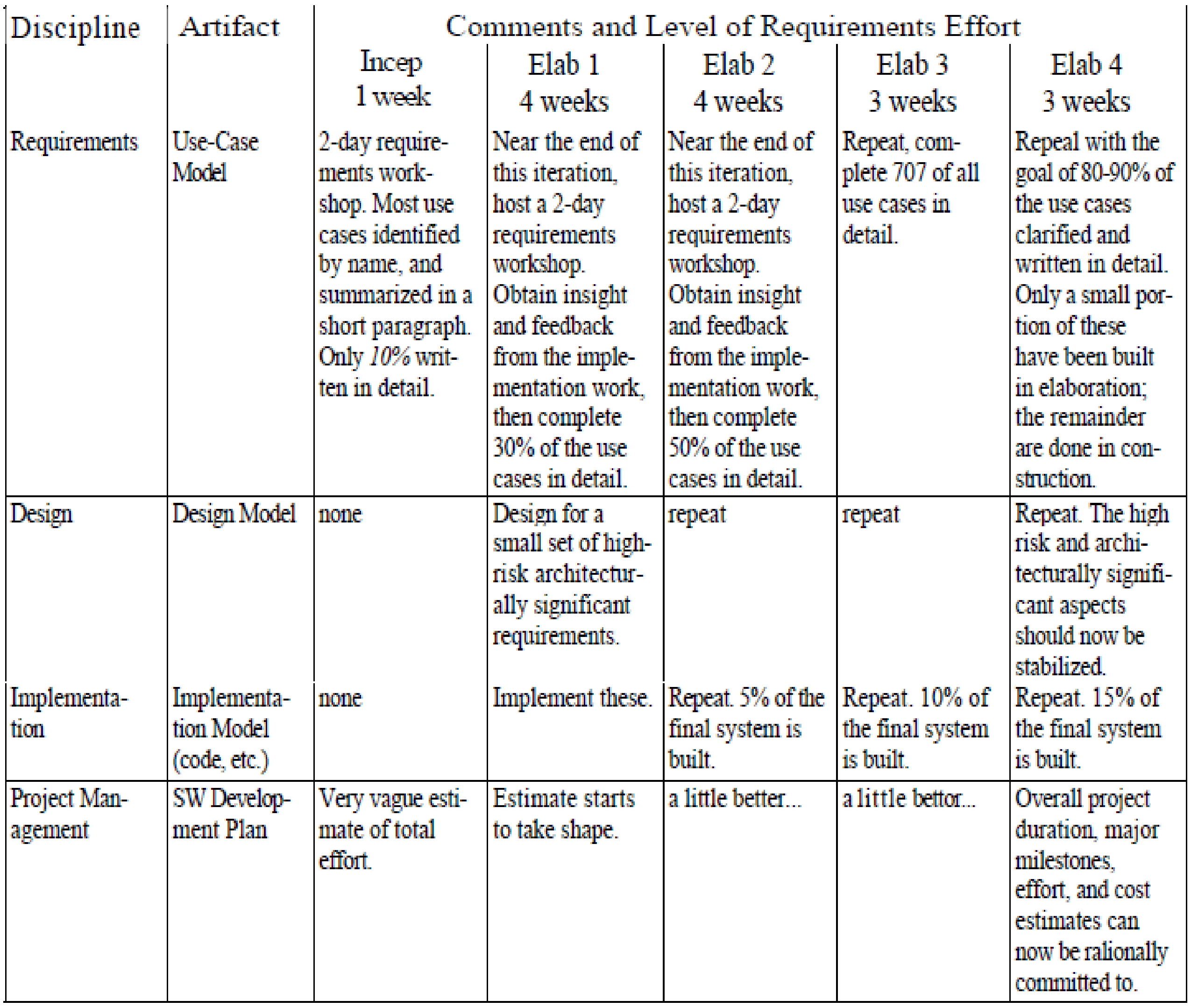
- Requirements are primarily recorded in use cases (the Use-Case Model) other requirements techniques (such as functions lists) are secondary, if used at all.
- Use cases are an important part of iterative planning. The work of iteration is. in part. Defined by choosing some use case scenarios, or entire 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.

The UP distinguishes between system and business use cases. **System use cases**, such as *Process Sale*. They are created in the Requirements discipline, and are part of the Use-Case Model.

Business use cases are less commonly written. If done, they are created in the Business Modeling discipline as part of a large-scale business process re-engineering effort, or to help understand the context of a new system in the business. They describe a sequence of actions of a business as a whole to fulfill a goal of a **business actor** (an actor in the business environment, such as a customer or supplier). For example, in a restaurant, one business use case is *Serve a Meal.*

Use Cases and Requirements Specification Across the Iterations:

The timing and level of effort of requirements specification across the iterations. Table 6.1 presents a sample (not a recipe) which communicates the UP strategy of how requirements are developed. Note that a technical team starts building the production core of the system when only perhaps 10% of the requirements are detailed, and in fact, there is a deliberate delay in continuing with concerted requirements work until near the end of the first elaboration iteration. This is the key difference in iterative development to a waterfall process: Production-quality development of the core of a system starts quickly, long before all the requirements are known.

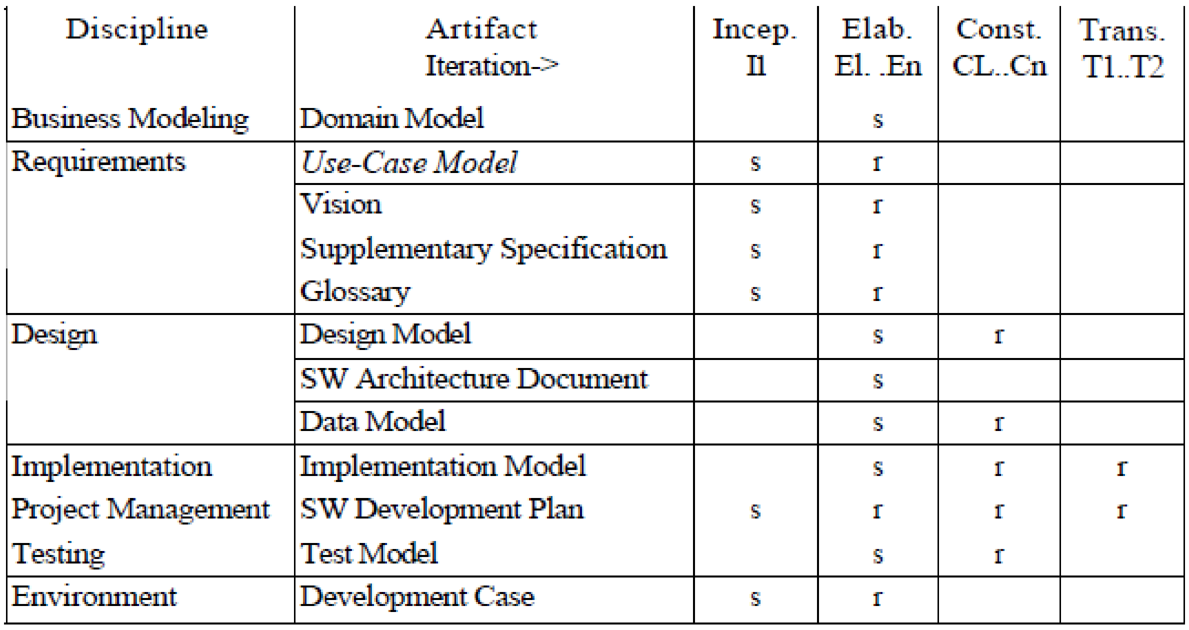


Discipline	Artifact	Comments and Level of Requirements Effort				
		Incep 1 week	Elab 1 4 weeks	Elab 2 4 weeks	Elab 3 3 weeks	Elab 4 3 weeks
Requirements	Use-Case Model	2-day requirements workshop. Most use cases identified by name, and summarized in a short paragraph. Only *10%* written in detail.	Near 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 707 of all use cases in detail.	Repeal 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 bettor...	Overall project duration, major milestones, effort, and cost estimates can now be rationally committed to.

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." That is, the act of building software rapidly surfaces assumptions and questions that need clarification.

Timing of UP Artifact Creation:

The Use-Case Model is started in inception, with perhaps only 10% of the 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 Iteration->	Incep. I1	Elab. E1..En	Const. C1..Cn	Trans. 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		
	Data Model		s	r	
Implementation	Implementation Model		s	r	r
Project Management	SW Development Plan	s	r	r	r
Testing	Test Model		s	r	
Environment	Development Case	s	r		

Sample UP artifacts and timing. s – Start r – refine.

Use Cases Within 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 user goals (and thus, user goal level use cases) are identified, including *Process Sale, Handle Returns,* and so on. 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, or which 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 a small sample of interesting use cases. Perhaps this means two use cases: *Process Sale* and *Handle Returns.*

A requirements management tool that integrates with a word processor is used for the writing, and the work is displayed via a projector while the team collaborates on the analysis and writing. The *Stakeholders and Interests* lists are written for these use cases, to discover more subtle (and perhaps costly) functional and key non-function requirements. Or system qualities. Such as for reliability or throughput. The analysis goal is not to exhaustively complete the use cases, but spend a few hours to obtain some insight.

The project sponsor needs to decide if the project is worth significant investigation (that is, the elaboration phase). The inception work is not meant to do that investigation, but to obtain low-fidelity (and admittedly error-prone) insights regarding scope, risk, effort, technical feasibility, and business case, in order to decide to move forward, where to start if they do, or if to stop. Perhaps the NextGen project inception step lasts five days. The combination of the two day requirements workshop and its brief use case analysis, and other investigation during the week, lead to the decision to continue on to an elaboration step for the system.

Use Cases With in Elaboration

The following discussion expands on the information in Table. 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 adap-tively 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 With in Construction

The construction step 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 will still be some minor use case writing and perhaps requirements workshops, but much less so than in elaboration. By this step, the majority of core functional and non-functional requirements should have iteratively and adaptively stabilized. That does not mean to imply requirements are frozen or investigation finished, but the degree of change is much lower.

UP Artifacts and Process Context

Use cases influence many UP Artifacts:

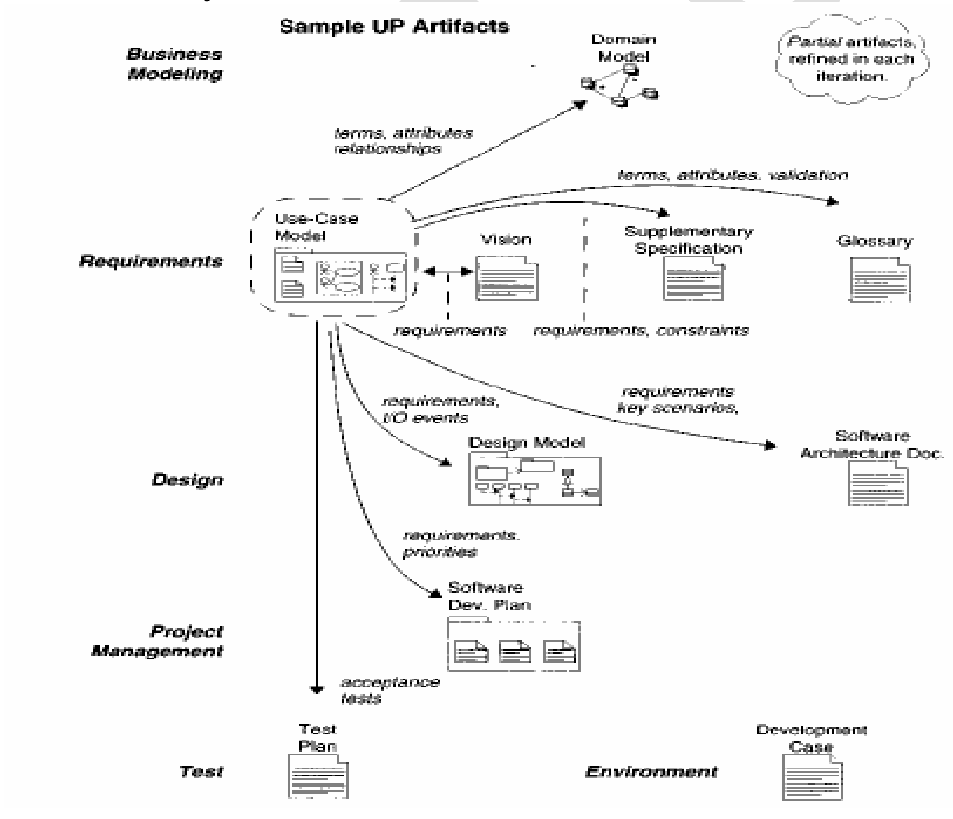
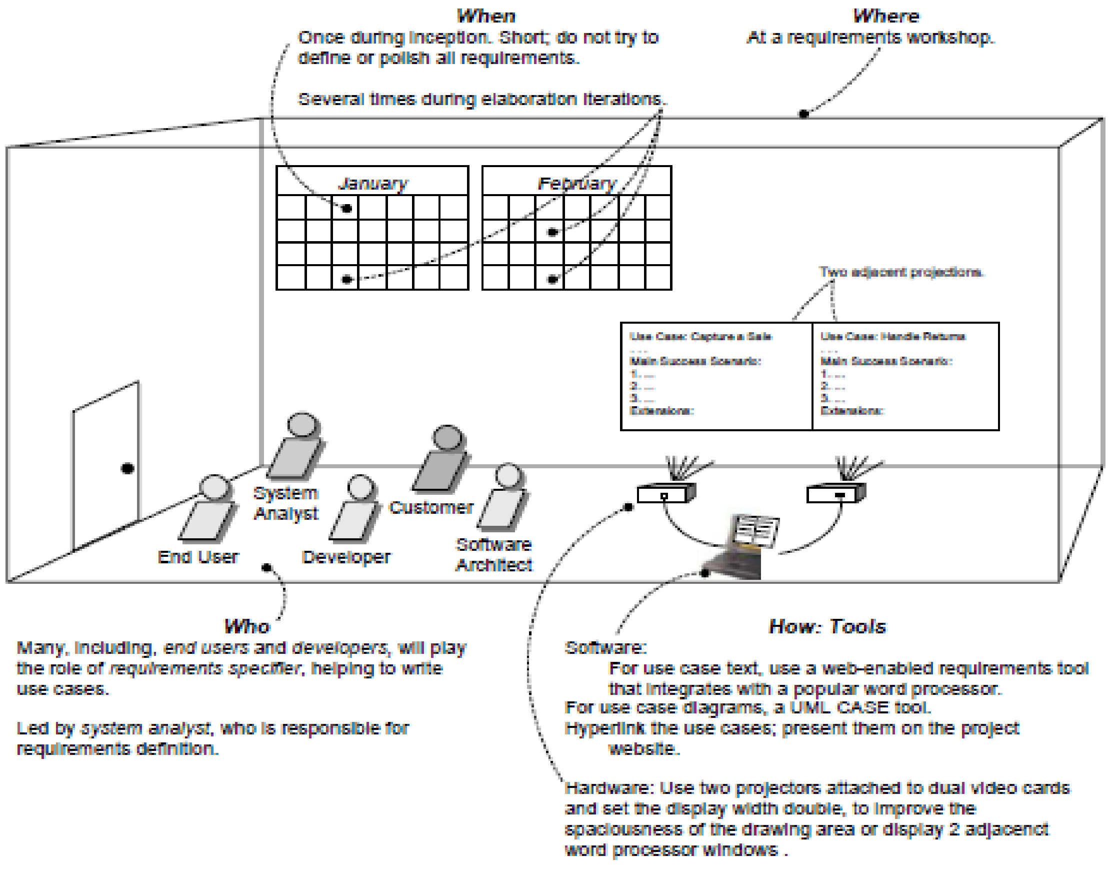


Figure 6.5 Sample UP artifact influence.

In the UP, use case work is a requirements discipline activity which could be initiated during a requirements workshop. Figure offers suggestions on the time and space for doing this work.


When
Once during inception. Short; do not try to define or polish all requirements.
Several times during elaboration iterations.
Where
At a requirements workshop.
January
February
Two adjacent projections.
Use Case: Capture a Sale
Main Success Scenario:
1. ...
2. ...
3. ...
Extensions:
Use Case: Handle Returns
Main Success Scenario:
1. ...
2. ...
3. ...
Extensions:
System Analyst
Customer
End User
Developer
Software Architect
Who
Many, including, end users and developers, will play the role of requirements specifier, helping to write use cases.
Led by system analyst, who is responsible for requirements definition.
How: Tools
Software:
For use case text, use a web-enabled requirements tool that integrates with a popular word processor.
For use case diagrams, a UML CASE tool.
Hyperlink the use cases; present them on the project website.
Hardware: Use two projectors attached to dual video cards and set the display width double, to improve the spaciousness of the drawing area or display 2 adjacenct word processor windows .

<u>Other Requirement Artifacts With in the UP</u>

As in the prior use case, Table summarizes a sample of artifacts and their timing. All requirements artifacts are started in inception, and primarily worked on through elaboration.

Discipline	Artifact Iteration->	Incep. I1	Elab. E1. .En	Const. C1..Cn	Trans. 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		
	Data Model		s	r	
Implementation	Implementation Model		s	r	r
Project Management	SW Development Plan	s	r	r	r
Testing	Test Model		s	r	
Environment	Development Case	s	r		

Inception:

It should not be the case that these requirements artifacts are finalized in the inception phase. Indeed, they will barely be started. Stakeholders need to decide if the project is worth serious investigation; that real investigation occurs during elaboration, not inception. During inception, the Vision summarizes the project idea in a form to help decision makers determine if it is worth continuing, and where to start.

Since most requirements work occurs during elaboration, the Supplementary Specification should be only lightly developed during inception, highlighting noteworthy quality attributes (for example, the NextGen POS must have recov-erability when external services fail) that expose major risks and challenges. Input into these artifacts could be generated during an inception phase requirements workshop, both through explicit consideration of its topics, and indirectly via use case analysis. Draft, readable artifacts will not get written in the workshop, but afterwards by the system analyst.

Elaboration

Through the elaboration iterations, the "vision" and the Vision are refined, based upon feedback from incrementally building parts of the system, adapting, and multiple requirements workshops over several development iterations.

Through ongoing requirements investigation and iterative development, the other requirements will become more clear and can be recorded in the SS. The quality attributes (for example, reliability) identified in the SS will be key drivers in shaping the core architecture that is designed and programmed during elaboration. They may also be key risk factors that influence what gets worked on in early iterations. For example, the NextGen POS quality requirement of client-side recoverability if external components fail will be explored during elaboration. The majority of terms will be discovered and elaborated in the Glossary during this phase.

By the end of elaboration, it is feasible to have use cases, a Supplementary Specification, and a Vision that reasonably reflects the stabilized major features and other requirements to be completed for delivery. Nevertheless, the Supplementary Specification and Vision are not something to freeze and "sign off" on as a fixed specification; adaptation not rigidity.is *a* core value of iterative development and the UP.

To clarify this "frozen sign off" comment: It is perfectly sensible.at the end of elaboration.to form an agreement with stakeholders about what will be done in the remainder of the project, and to make commitments (perhaps contractual) regarding requirements and schedule. At some point (the end of elaboration, in the UP), we need a reliable idea of "what, how much, and when." In that sense, a formal agreement on the requirements is normal and expected. It is also necessary to have a change control process (one of the explicit best practice in the UP) so that changes in requirements are formally considered and approved, rather than chaotic and uncontrolled change.

Rather, several ideas are implied by the "frozen sign off" comment:

- . In iterative development and the UP it is understood that no matter how much due diligence is given to requirements specification, some change is inevitable, and should be acceptable. This change could be a late-breaking opportunistic improvement in the system that gives its owners a competitive advantage, or change due to improved insight.
- . In iterative development, it is a core value to have continual engagement by the stakeholders to evaluate, provide feedback, and steer the project as they really want it. It does not benefit stakeholders to "wash their hands" of attentive engagement by signing off on a frozen set of requirements and waiting for the finished product, because they will seldom get what they really needed.

Construction

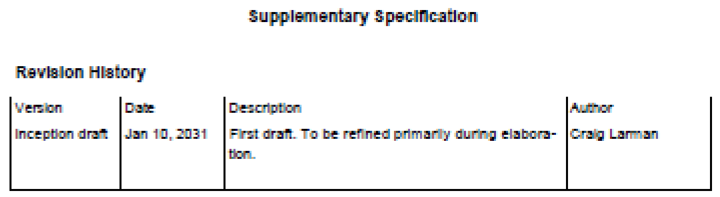
By construction, the major requirements.both functional and otherwise. should be stabilized.not finalized, but settled down to minor pertubation. Therefore, the SS and Vision are unlikely to experience much change in this phase.

Writing requirements for the case study in the use case model

NextGen POS Examples

The purpose of the following examples is not to present an exhaustive Vision, Glossary, or Supplementary Specification, as some of the sections. although useful for a project.are not relevant to the learning objectives.1 The book’s goal is core skills in object design, use case requirements analysis, and object-oriented analysis, not POS problems or Vision statements. Therefore, only some sections are briefly touched upon in order to make connections between prior and future work, highlight noteworthy issues, provide a feel for the contents, and move forward quickly.

NextGen Example: (Partial) Supplementary Specification



Supplementary Specification

Revision History

Version	Date	Description	Author
Inception draft	Jan 10, 2031	First draft. To be refined primarily during elaboration.	Craig Larman

Introduction

This document is the repository of all NextGen POS requirements not captured in the use cases.

Functionality

(Functionality common across many use cases)

Logging and Error Handling Log all errors to persistent storage. ***Pluggable Business Rules***

At various scenario points of several use cases (to be defined) support the ability to customize the functionality of the system with a set of arbitrary rules that execute at that point or event.

Security

All usage requires user authentication.

Usability

Human Factors

The customer will be able to see a large-monitor display of the POS.Therefore:

- Text should be easily visible from 1 meter.
- Avoid colors associated with common forms of color blindness.

Speed, ease, and error-free processing are paramount in sales processing, as the buyer wishes to leave quickly, or they perceive the purchasing experience (and seller) as less positive. The cashier is often looking at the customer or items, not the computer display. Therefore, signals and warnings should be conveyed with sound rather than only via graphics.

Reliability

Recoverability

If there is failure to use external services (payment authorizer, accounting system, ...) try to solve with a local solution (e.g., store and forward) in order to still complete a sale. Much more analysis is needed here...

Performance

As mentioned under human factors, buyers want to complete sales processing very quickly. One potential bottleneck is external payment authorization. Our goal is to achieve authorization in less than 1 minute, 90% of the time.

Supportability

Adaptability

Different customers of the NextGen POS have unique business rule and processing needs while processing a sale. Therefore, at several defined points in the scenario (for example, when a new sale is initiated, when a new line item is added) pluggable business rule will be enabled.

Configurability

Different customers desire varying network configurations for their POS systems, such as thick versus thin clients, two-tier versus N-tier physical layers, and so forth. In addition, they desire the ability to modify these configurations, to reflect their changing business and performance needs. Therefore, the system will be somewhat configurable to reflect these needs. Much more analysis is needed in this area to discover the areas and degree of flexibility, and the effort to achieve it.

Implementation Constraints

NextGen leadership insists on a Java technologies solution, predicting this will improve long-term porting and supportability, in addition to ease of development.

Purchased Components

Tax calculator. Must support pluggable calculators for different countries.

Free Open Source Components

In general, we recommend maximizing the use of free Java technology open source components on this project. Although it is premature to definitively design and choose components, we suggest the following as likely candidates:

- JLog logging framework

Interfaces

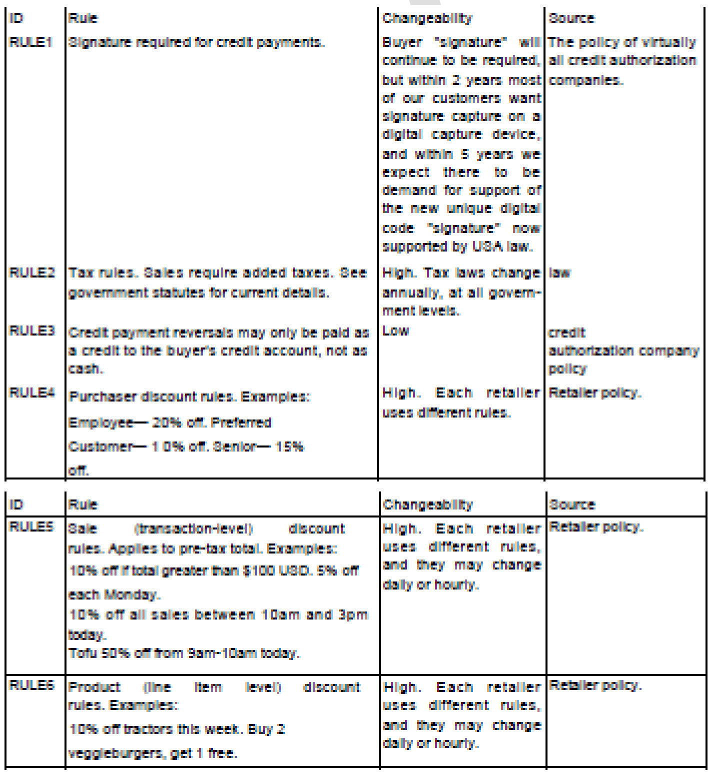
Noteworthy Hardware and Interfaces

- Touch screen monitor (this is perceived by operating systems as a regular monitor, and the touch gestures as mouse events)
- Barcode laser scanner (these normally attach to a special keyboard, and the scanned input is per ceived in software as keystrokes)
- Receipt printer
- Credit/debit card reader
- Signature reader (but not in release 1)

Software Interfaces

For most external collaborating systems (tax calculator, accounting, inventory, ...) we need to be able to plug in varying systems and thus varying interfaces.

Domain (Business) Rules



ID	Rule	Changeability	Source
RULE1	Signature required for credit payments.	Buyer "signature" will continue to be required, but within 2 years most of our customers want signature capture on a digital capture device, and within 5 years we expect there to be demand for support of the new unique digital code "signature" now supported by USA law.	The policy of virtually all credit authorization companies.
RULE2	Tax rules. Sales require added taxes. See government statutes for current details.	High. Tax laws change annually, at all govern-ment levels.	law
RULE3	Credit payment reversals may only be paid as a credit to the buyer's credit account, not as cash.	Low	credit authorization company policy
RULE4	Purchaser discount rules. Examples: Employee— 20% off. Preferred Customer— 1 0% off. Senior— 15% off.	High. Each retailer uses different rules.	Retailer policy.
RULE5	Sale (transaction-level) discount rules. Applies to pre-tax total. Examples: 10% off if total greater than $100 USD. 5% off each Monday. 10% off all sales between 10am and 3pm today. Tofu 50% off from 9am-10am today.	High. Each retailer uses different rules, and they may change daily or hourly.	Retailer policy.
RULE6	Product (line item level) discount rules. Examples: 10% off tractors this week. Buy 2 veggieburgers, get 1 free.	High. Each retailer uses different rules, and they may change daily or hourly.	Retailer policy.

Legal Issues

We recommend some open source components if their licensing restrictions can be resolved to allow resale of products that include open source software.

All tax rules must, by law, be applied during sales. Note that these can change frequently.

Information in Domains of Interest

Pricing

In addition to the pricing rules described in the domain rules section, note that products have an *original price,* and optionally a *permanent markdown price.* A product's price (before further discounts) is the permanent markdown price, if present. Organizations maintain the original price even if there is a permanent markdown price, for accounting and tax reasons.

Credit and Debit Payment Handling

When an electronic credit or debit payment is approved by a payment authorization service, they are responsible for paying the seller, not the buyer. Consequently, for each payment, the seller needs to record monies owing in their accounts receivable, from the authorization service. Usually on a nightly basis, the authorization service will perform an electronic funds transfer to the seller's account for the daily total owing, less a (small) per transaction fee that the service charges.

Sales Tax

Sales tax calculations can be very complex, and regularly change in response to legislation at all levels of government. Therefore, delegating tax calculations to third-party calculator software (of which there are several available) is advisable. Tax may be owing to city, region, state, and national bodies. Some items may be tax exempt without qualification, or exempt depending on the buyer or target recipient (for example, a farmer or a child).

Item Identifiers: UPCs, EANs, SKUs, Bar Codes, and Bar Code Readers

The NextGen POS needs to support various item identifier schemes. UPCs (Universal Product Codes), EANs (European Article Numbering) and SKUs (Stock Keeping Units) are three common identifier systems for products that are sold. Japanese Article Numbers (JANs) are a kind of EAN version. SKUs are completely arbitrary identifiers defined by the retailer. However, UPCs and EANs have a standards and regulatory component.

***