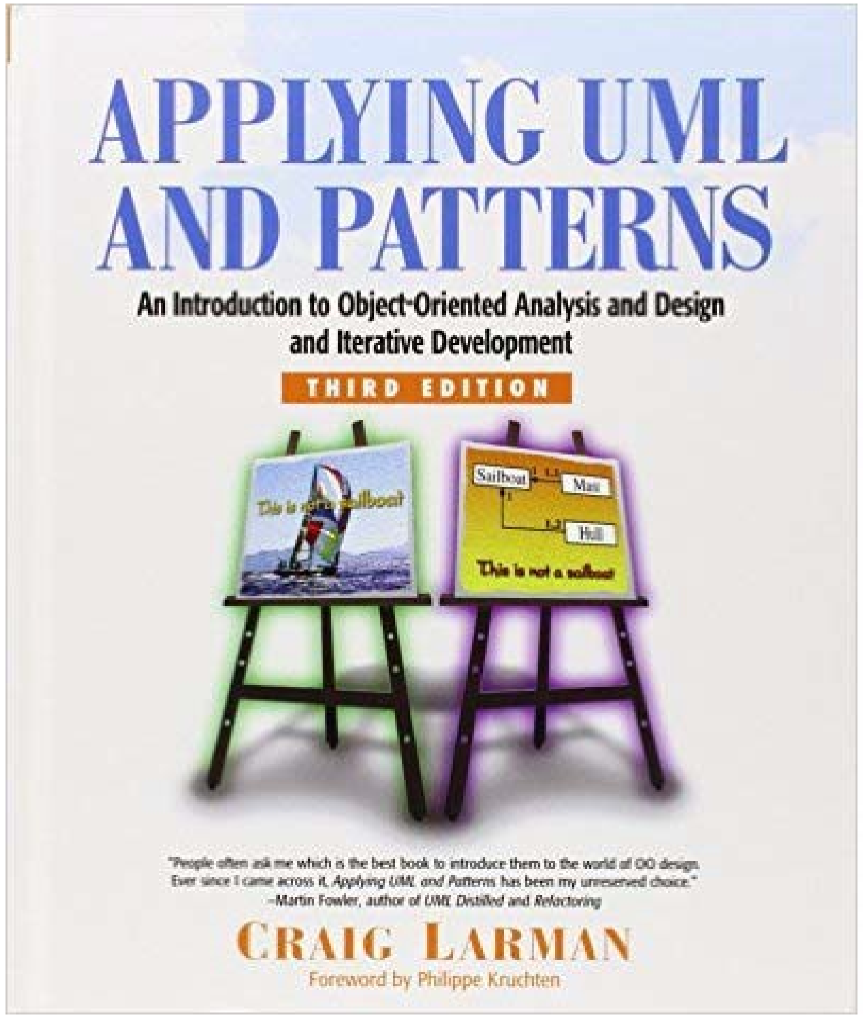
Object-Oriented Analysis and Design
Inception

Yunho Kim
Hanyang Univ.

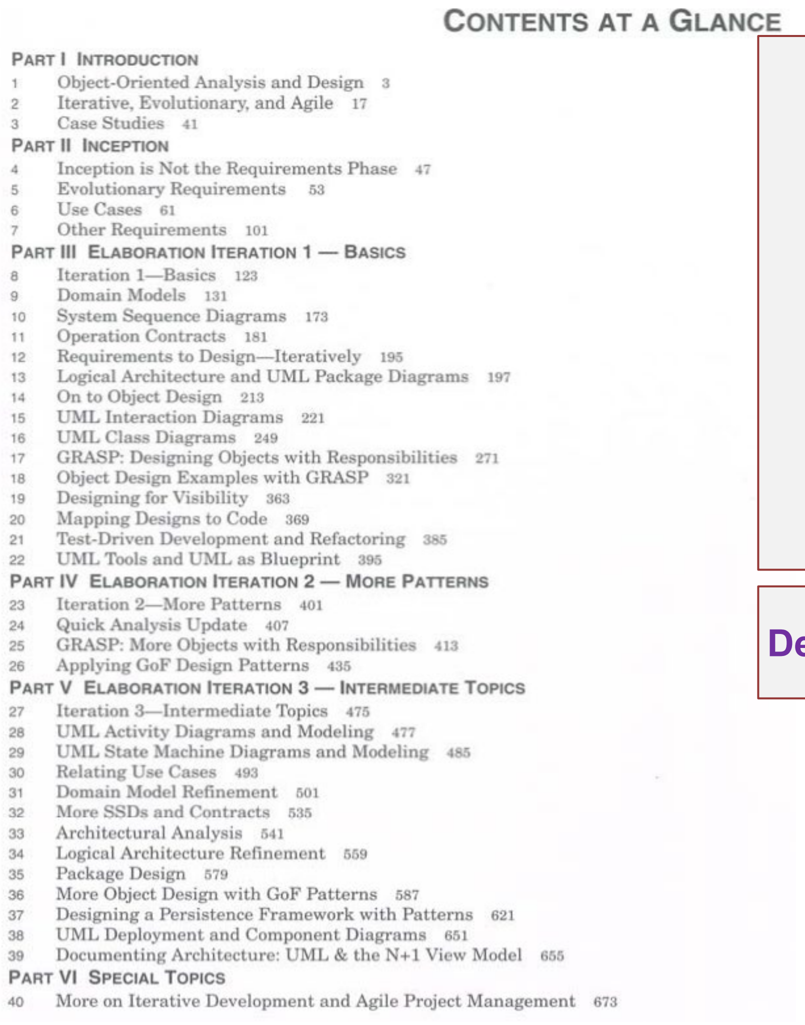
What We Cover

- ✧ Summary of OOAD
- ✧ UP Inception
- ✧ UP Elaboration
  - ▪ OO Analysis
  - ▪ OO Design

Text and Contents



APPLYING UML AND PATTERNS
An Introduction to Object-Oriented Analysis and Design and Iterative Development
THIRD EDITION
This is not a sailboat
Sailboat
Mast
Hull
This is not a sailboat
"People often ask me which is the best book to introduce them to the world of OO design. Ever since I came across it, Applying UML and Patterns has been my unreserved choice." –Martin Fowler, author of UML Distilled and Refactoring
CRAIG LARMAN
Foreword by Philippe Kruchten


CONTENTS AT A GLANCE

OOAD

Design Patterns

Object-Oriented Analysis and Design

- ✧ **Object-Oriented Analysis (OOA)**
  - Discover the domain <u>concepts/objects</u> (the objects of the problem domain)

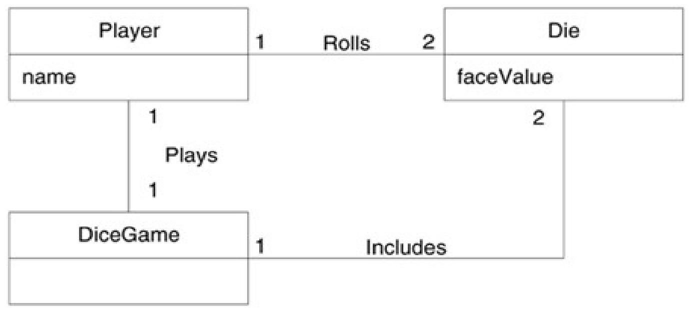
- ✧ **Object-Oriented Design (OOD)**
  - Define <u>software objects</u> (static)
  - Define <u>how they collaborate</u> to fulfill the requirements (dynamic)

An OOAD Example - Dice Game

Define use cases → Define domain model → Define interaction diagrams → Define design class diagrams

OOA

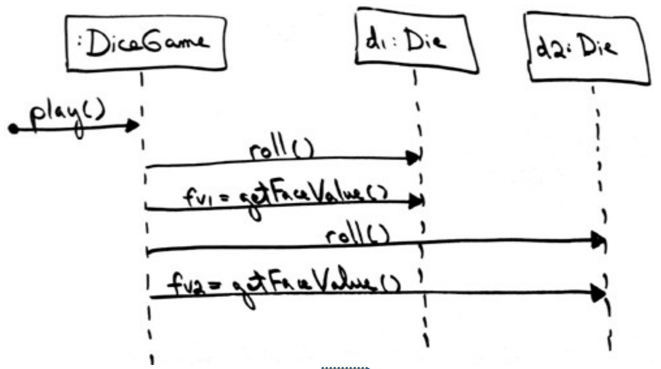
Use Case : Play a Dice Game
- Player requests to roll the dice.
- System presents results.
- If the dice's face value totals seven, player wins; otherwise, player loses.

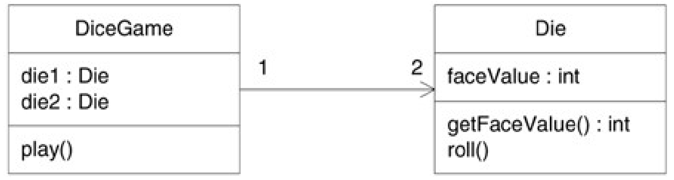


Domain Model

OOD

Interaction Diagram





Design Class Diagram

Software Development Process and the UP

✧ **Software development process**

- A **systematic approach** to building, deploying and possibly maintaining software

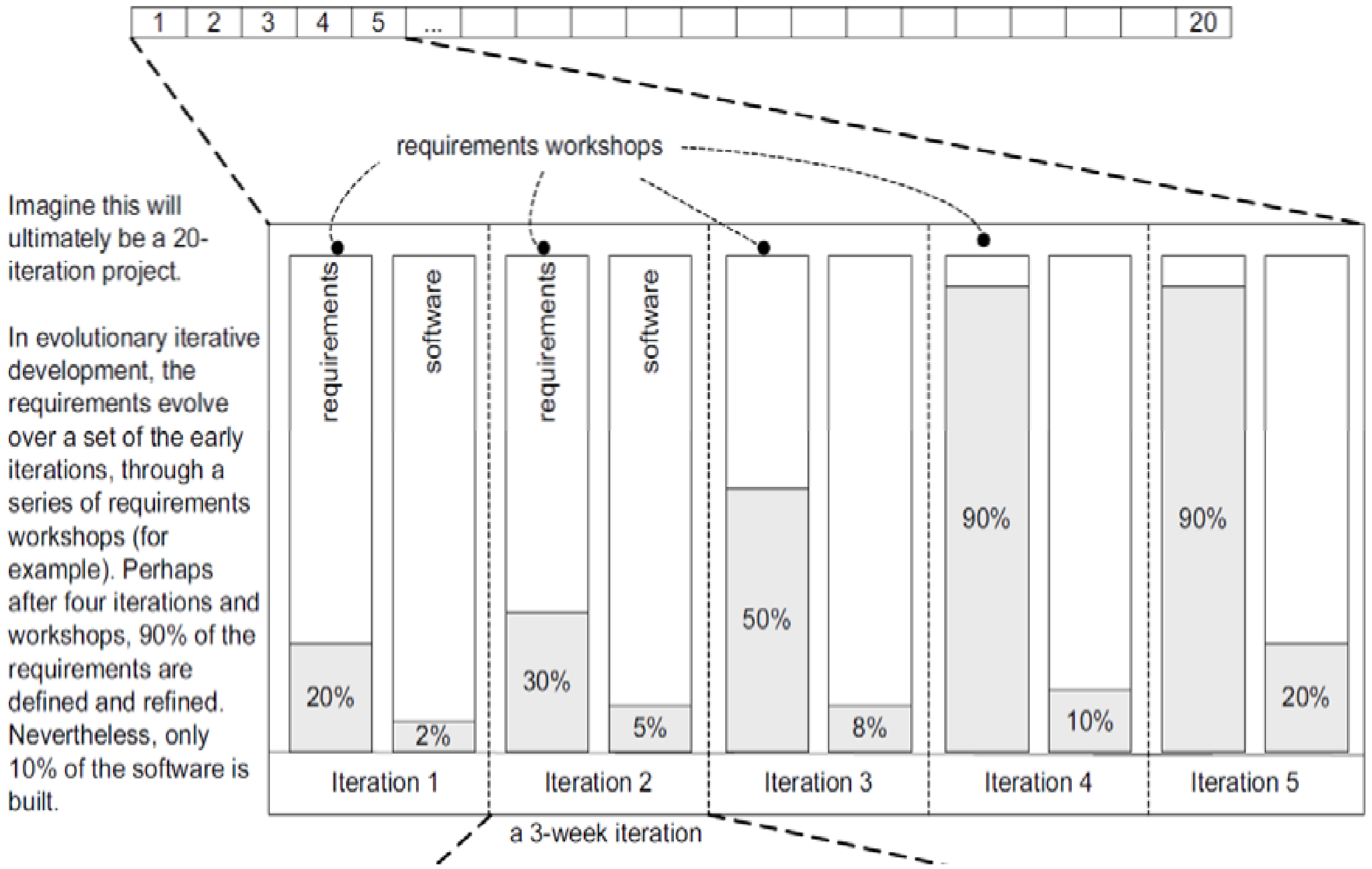
✧**Unified Process (UP)**: a popular iterative software development process for building object-oriented systems

- Iterative with fixed-length iterations (mini waterfalls of about **3 weeks**)
- Inspired from Agile (*i.e.,* opposite from waterfall)
- Flexible (can be combined with practices from other OO processes)
- Widely used in industry for developing OO software

Risk-Driven and Client-Driven Iterative Planning

- ✧ The **UP** encourages a combination of **risk-driven** and **client-driven iterative planning**.
  - To identify and drive down the high risks, and
  - To build visible features that clients care most about.

- ✧ **Risk-driven** iterative development includes more specifically the practice of **architecture-centric** iterative development.
  - Early iterations focus on building, testing, and stabilizing the core architecture.


1
2
3
4
5
...
20
requirements workshops
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.
requirements
software
20%
2%
30%
5%
50%
8%
90%
10%
90%
20%
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
a 3-week iteration

The UP Practices

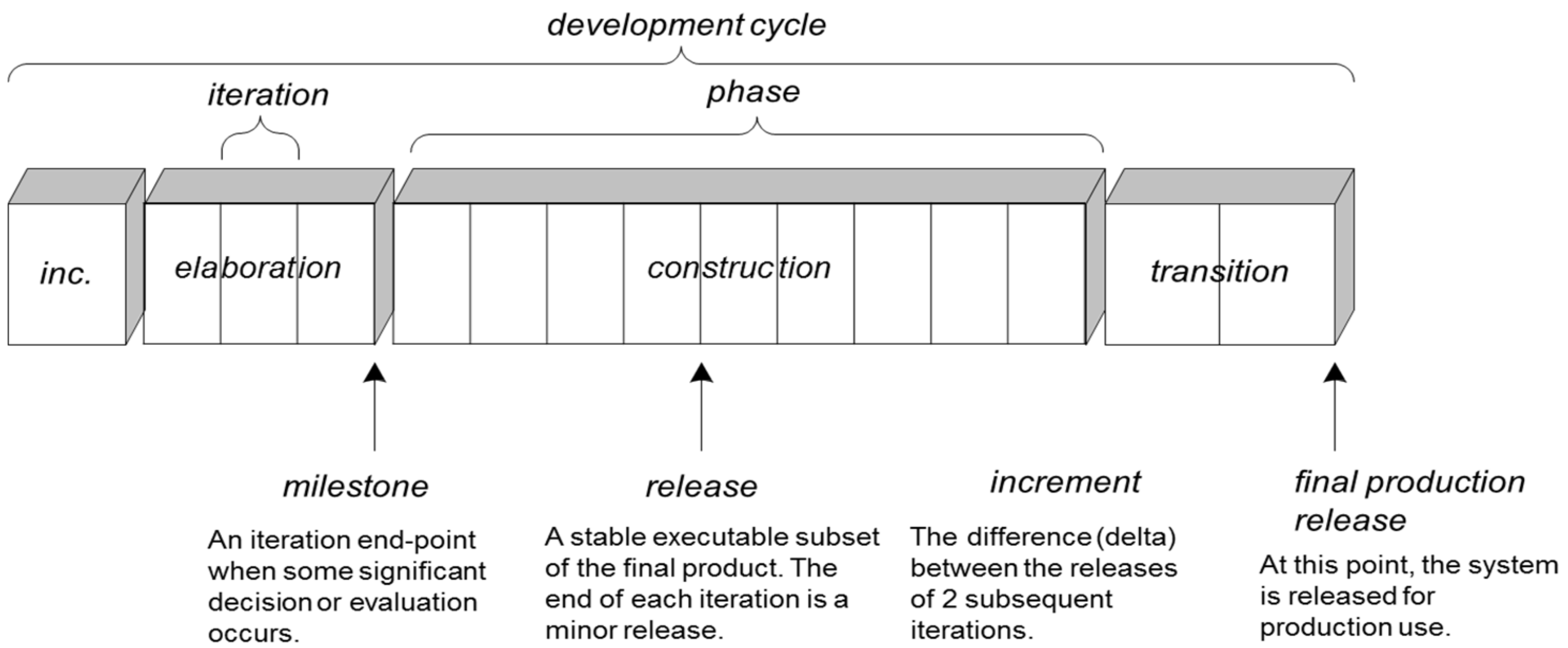
- ✧ The central idea to **UP practices** :
  - A short timeboxed **iterative**, **evolutionary** and **adaptive** development

- ✧ Additional **best practices** and **key concepts**:
  - Tackle high-risk and high-value issues in early iterations **(→ Risk-driven, Client-driven)**
  - Continuously engage users for evaluation and feedback **(→ Client-driven)**
  - Build a cohesive, core architecture in early iterations **(→ Architecture-centric)**
  - Continuously verify quality; test early, often, and realistically
  - Apply use cases where appropriate
  - Do some visual modeling (with the UML)
  - Carefully manage requirements (configuration management)

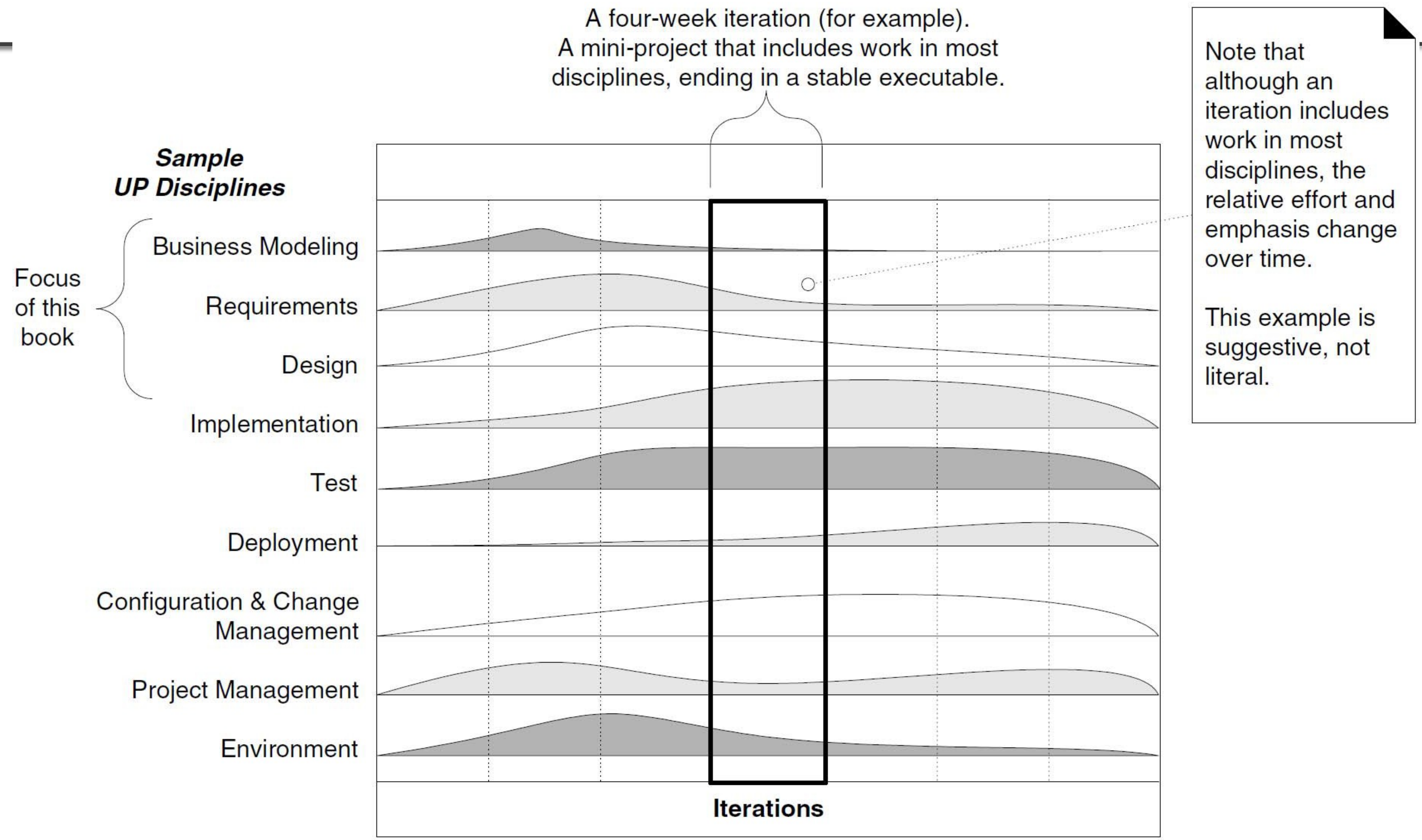
The UP Phases

✧ A UP project organizes the work and iterations across **4 major phases**:

1. **Inception** : approximate vision, business case, scope, vague cost estimates
2. **Elaboration** : refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates
3. **Construction** : iterative implementation of the remaining lower risk and easier elements, and preparation for deployment
4. **Transition** : beta tests, deployment

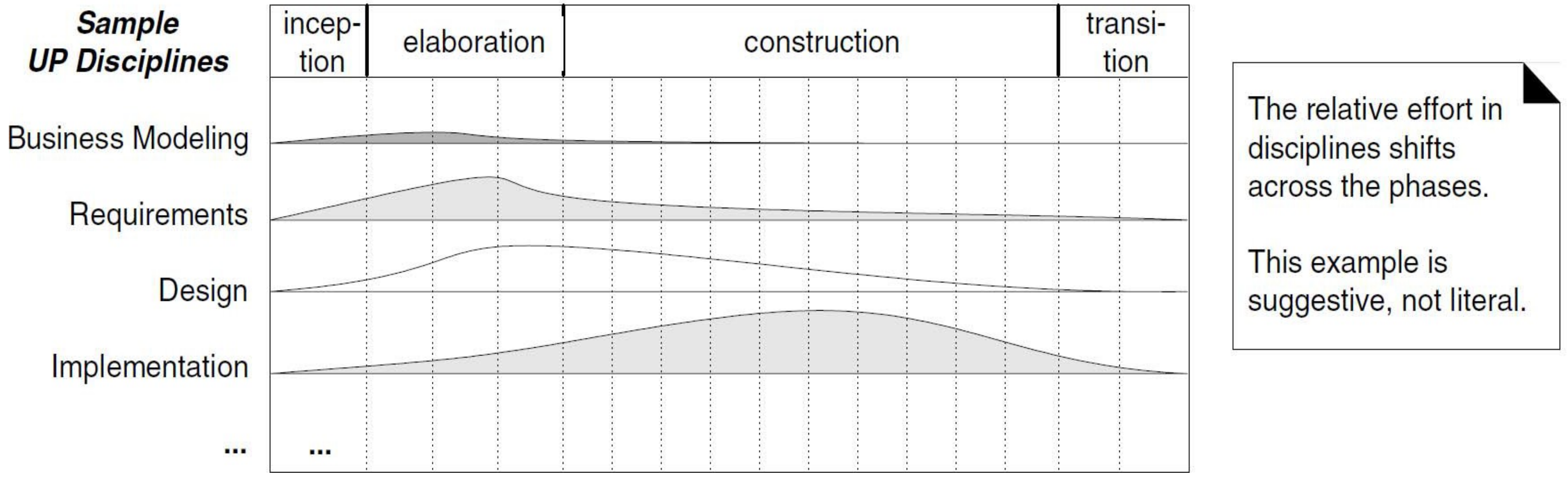

development cycle
iteration
phase
inc.
elaboration
construction
transition
milestone
An iteration end-point when some significant decision or evaluation occurs.
release
A stable executable subset of the final product. The end of each iteration is a minor release.
increment
The difference (delta) between the releases of 2 subsequent iterations.
final production release
At this point, the system is released for production use.

The UP Disciplines


A four-week iteration (for example).
A mini-project that includes work in most disciplines, ending in a stable executable.
Note that although an iteration includes work in most disciplines, the relative effort and emphasis change over time.
This example is suggestive, not literal.
Sample UP Disciplines
Focus of this book
Business Modeling
Requirements
Design
Implementation
Test
Deployment
Configuration & Change Management
Project Management
Environment
Iterations

Relationship Between the Disciplines and Phases

✧ The relative effort in disciplines **shifts** to across the phases.



✧Artifact : A general term for any work product

- Example: code, web graphics, database schema, text documents, diagrams, models and so on

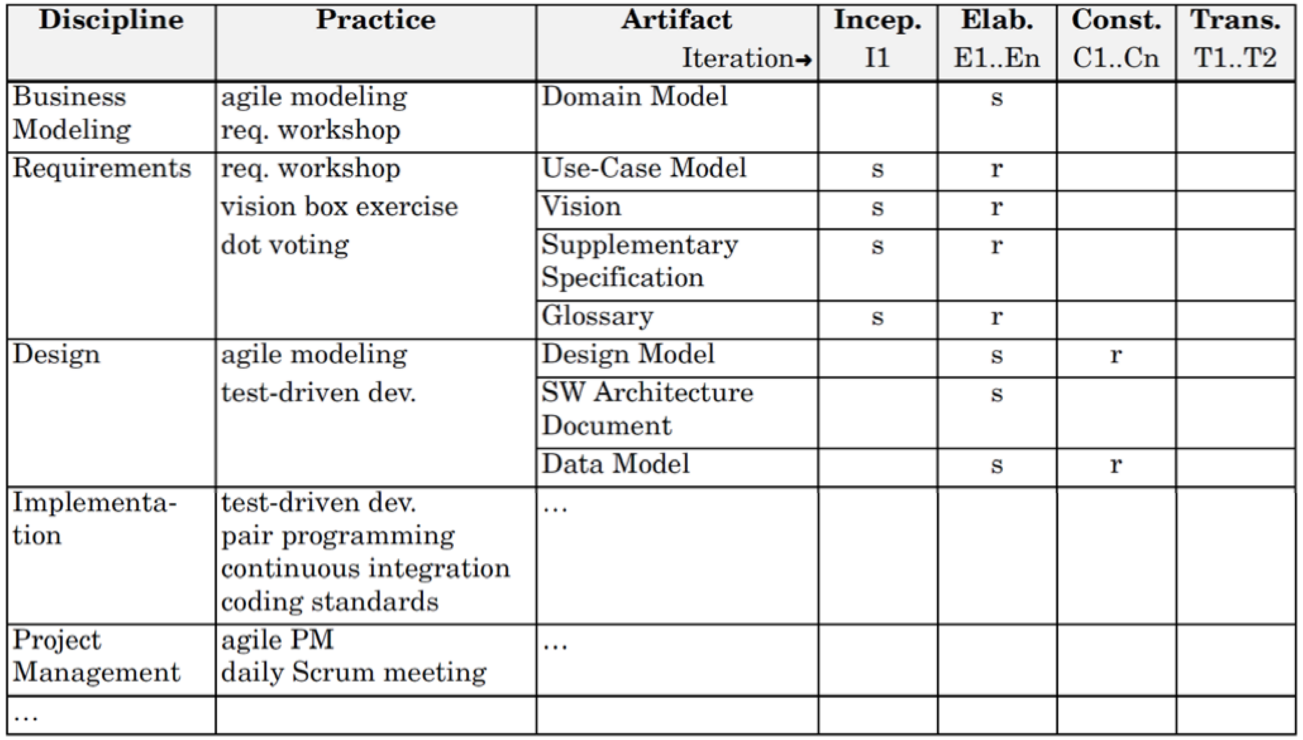
✧Discipline : A set of activities and related artifacts in one subject area

- Example: the activities within requirements analysis

The UP Development Case

✧ **Development Case**:

- An artifact in the Environment discipline
- Documenting the choice of practices and UP artifacts for a project
- For example, the development case for the NextGen POS case study :



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	
Implementa-tion	test-driven dev. pair programming continuous integration coding standards	...				
Project Management	agile PM daily Scrum meeting	...				
...						

Case One: The NextGen POS System

The first case study is the NextGen point-of-sale (POS) system. In this apparently straightforward problem domain, we shall see that there are interesting requirement and design problems to solve. In addition, it's a real problemgroups really do develop POS systems with object technologies.

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. These systems must be relatively fault-tolerant; that is, even if remote services are temporarily unavailable (such as the inventory system), they must still be capable of capturing sales and handling at least cash payments (so that the business is not crippled).

A POS system increasingly must support multiple and varied client-side terminals and interfaces. These include a thin-client Web browser terminal, a regular personal computer with something like a Java Swing graphical user interface, touch screen input, wireless PDAs, and so forth.

Furthermore, we are creating a commercial POS system that we will sell to different clients with disparate needs in terms of business rule processing. Each client will desire a unique set of logic to execute at certain predictable points in scenarios of using the system, such as when a new sale is initiated or when a new line item is added. Therefore, we will need a mechanism to provide this flexibility and customization.

Using an iterative development strategy, we are going to proceed through requirements, object-oriented analysis, design, and implementation.

Inception

- ✧ Inception is the initial **short** step that is used to establish a common vision and basic scope for the project
- ✧ Main questions that are often asked:
  - What is the overall vision and business case for the project?
  - Is it feasible?
  - Buy or build?
  - Rough cost estimate (order of magnitude)
  - Go, no go
- ✧ Inception should be short
  - **One week** for most projects
  - We do not define all of the requirements in Inception!
    - Perhaps a couple of example requirements, use cases

Inception

- ✧ **Goal**: Envision the project scope, vision, and business case.
- ✧ Is there a basic agreement among the stakeholders on the vision, and is it worth investing in a serious *investigation*?
  - ▪ Note *investigation* versus *development*
- ✧ There may be some simple UML diagrams, and even some basic coding for proof-of-concept prototypes to answer key questions

Evolutionary Requirements

- ✧ **Requirements** are **capabilities** and **conditions** to which **the system must conform**.
- ✧ **Requirement analysis** is to **find**, communicate and **organize** what is really needed, in a form that is clear both to clients and team members
- ✧ Since UP does not require all requirements to be defined up front, it requires careful *management* of requirements
  - ▪ “a systematic approach to finding, documenting, organizing, and tracking the changing requirements of the system”
- ✧ Key difference between Waterfall and UP: UP *embraces* requirements changes

Evolutionary Requirements

✧ How to find the requirements?

- Different methodologies do this in different ways; Requirements Workshops, Use Cases, etc.

✧ The UP encourages skillful elicitation (finding) via techniques such as

- writing use cases **with customers**,
- **requirements workshops** that include both developers and customers,
- **a demo of the results** of each iteration to the customers, to solicit feedback

Types and Categories of Requirements: FURPS+

✧ In the UP, requirements are categorized according to the **FURPS+ model**

- Functional – features, capabilities, security
- Usability – human factors, documentation
- Reliability – frequency of failure, recoverability, predictability
- Performance – response times, throughput, accuracy, availability, resource usage
- 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 : for example, a physical box
  - Legal : Licensing and so forth

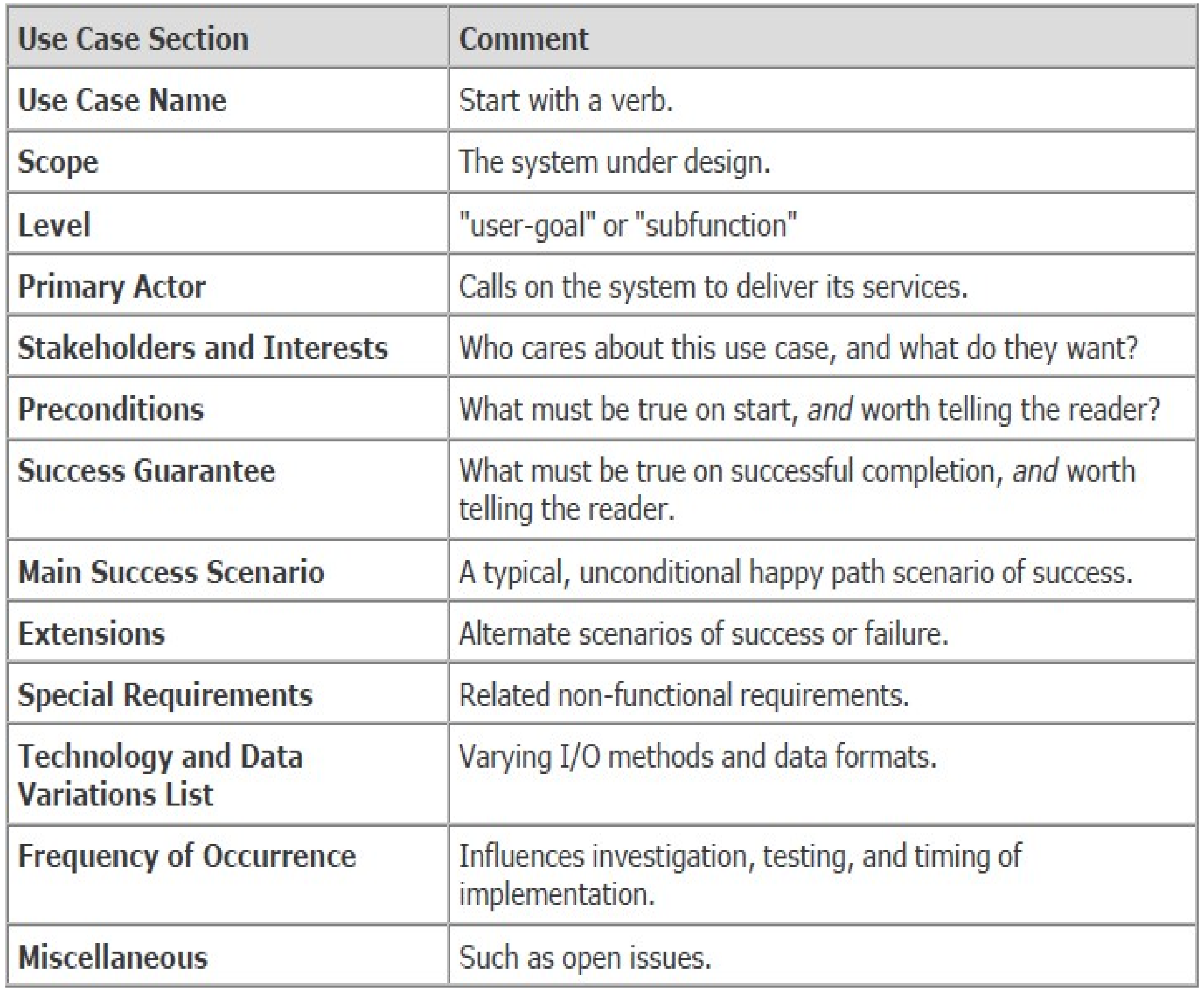
Requirements Organization: UP Artifacts

- The UP offers several requirements artifacts. (But, they are all optional.)
  - **Use-Case Model**
    - A set of typical scenarios of using a system
    - These are primarily for functional (behavioral) requirements.
  - **Supplementary Specification**
    - Basically, everything is not in the use cases.
    - This artifact is primarily for all non-functional requirements, such as performance or licensing.
    - It is also the place to record functional features not expressed (or expressible) as use cases; for example, a report generation.
  - Glossary
    - It defines noteworthy terms.
  - Vision
    - A short executive overview document for quickly learning the project's big ideas.
  - Business Rules
    - It typically describe requirements or policies that transcend one software project.

Use Cases

✧ **Use cases** are **text stories** of some actors using a system to meet goals.

- A mechanism to capture (analyzes) requirements
- An example (Brief format):
  - **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 a 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.
- Use case is not a diagram, but a text.



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, *and* worth telling the reader?
Success Guarantee	What must be true on successful completion, *and* 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.

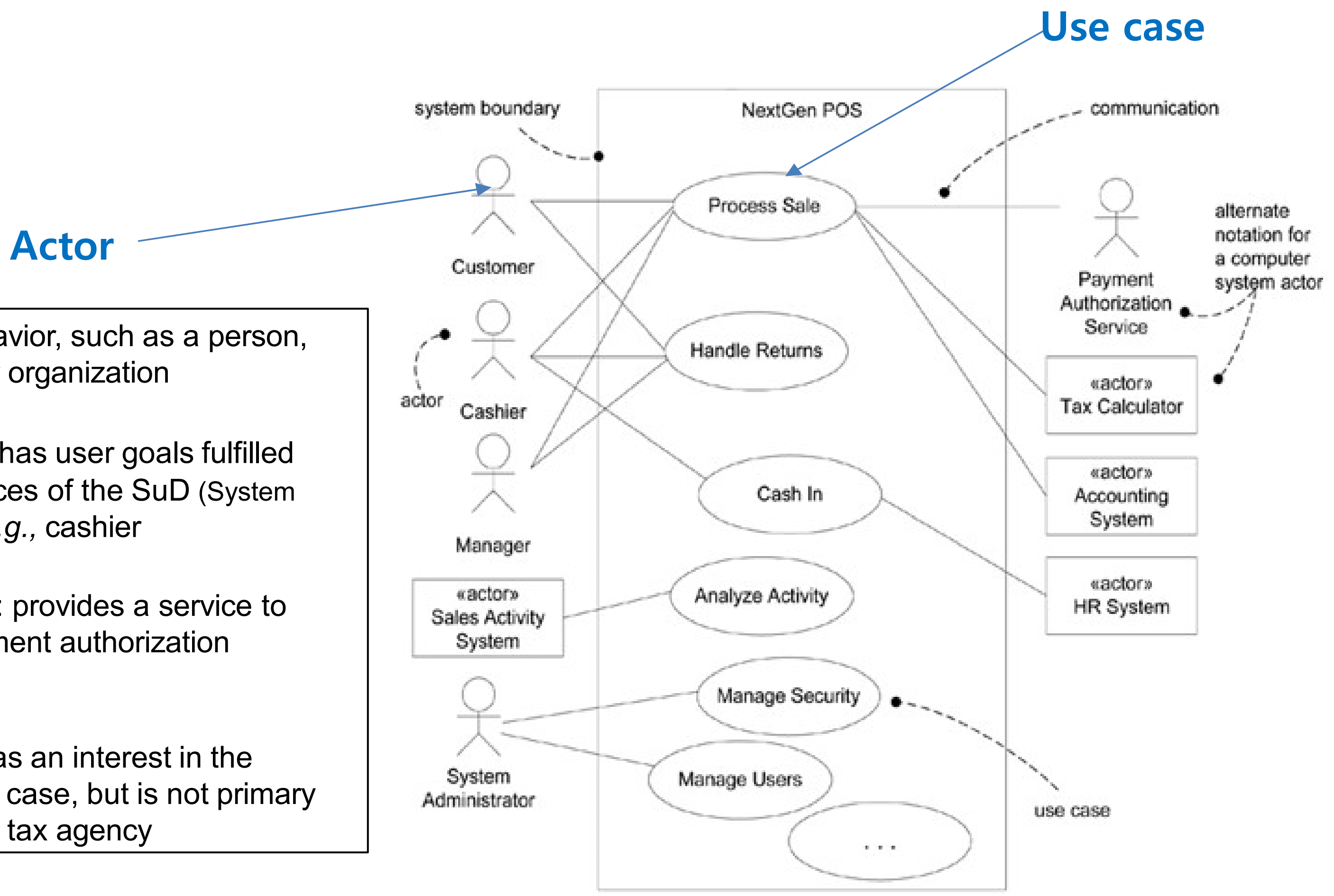
Use Case Diagram

✧ **Use case diagram** illustrates the name of use cases and actors, and the relationships between them.

- System context diagram
- A summary of all use cases

Something with behavior, such as a person, computer system, or organization

- **Primary Actor** : has user goals fulfilled through using services of the SuD (System Under Discussion) , *e.g.,* cashier
- **Supporting Actor** : provides a service to the SuD, *e.g.,* payment authorization service
- **Offstage Actor** : has an interest in the behavior of the use case, but is not primary or supporting, *e.g.,* tax agency



Are Use Cases Functional Requirements?

- Yes, **Use Cases are requirements**, primarily **functional (behavioral)** requirements.
  - "**F**" (functional or behavioral) in terms of **FURPS+** requirements types
  - Can also be used for other types.

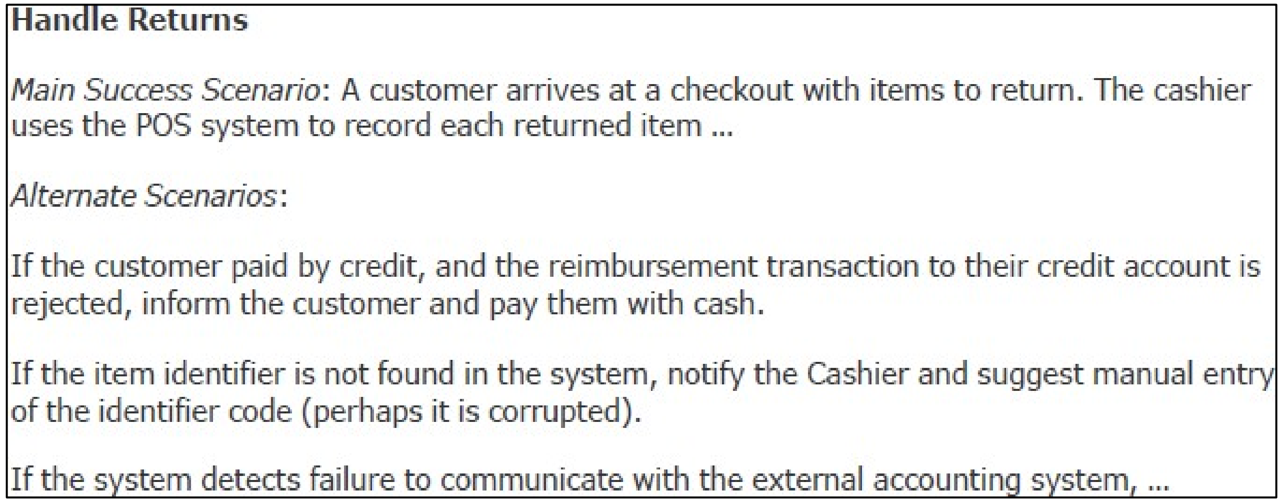
Three Common Use Case Formats

✧ Brief :

- Terse one paragraph summary, usually the main success scenario or a happy path

✧ Casual :

- Informal paragraph format.
- Multiple paragraphs that cover various scenarios.



Handle Returns

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 customer paid by credit, and the reimbursement transaction to their credit account is rejected, inform the customer and pay them with cash.

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 accounting system, ...

✧ Fully Dressed :

- Includes all steps, variations and supporting sections (e.g., preconditions)

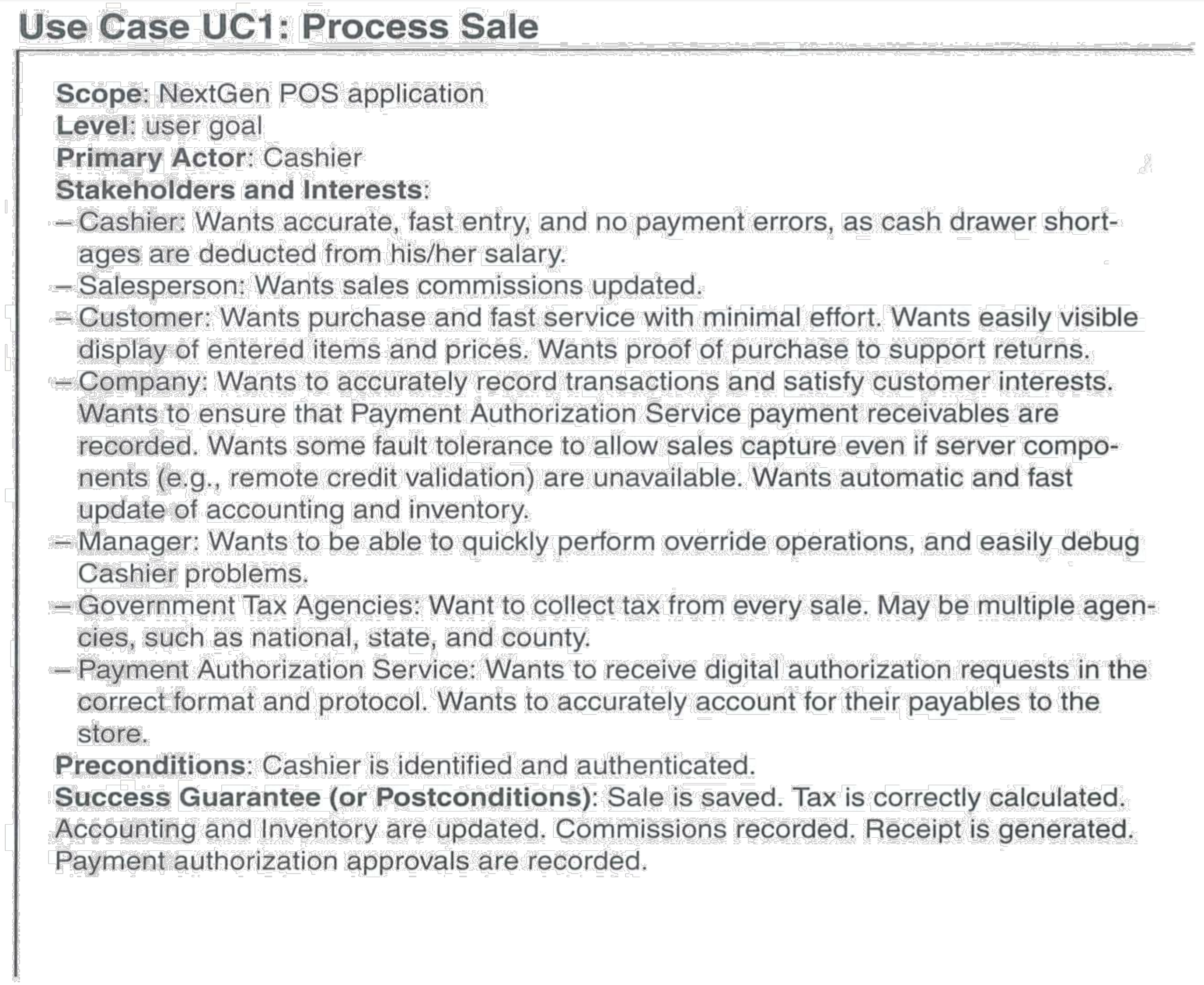
Use Case Template (Fully Dressed) (1/2)

Use Case Section	Comment
Use Case Name	Starts with verb, unique, sometimes number
Scope	Identifies the system under design
Level	User-goal level, but may be subfunction if these substeps are used by many use cases
Primary Actor	Actor that calls upon the system to fulfill a goal
Stakeholders and Interests List	Very important – lists all stakeholders in the scenario, and what they expect the system to do. Will identify the behaviors.
Preconditions, Success Guarantees	Conditions that are relevant and considered true at the start of the use case; what must be true upon completion of the scenario

Use Case Template (Fully Dressed) (2/2)

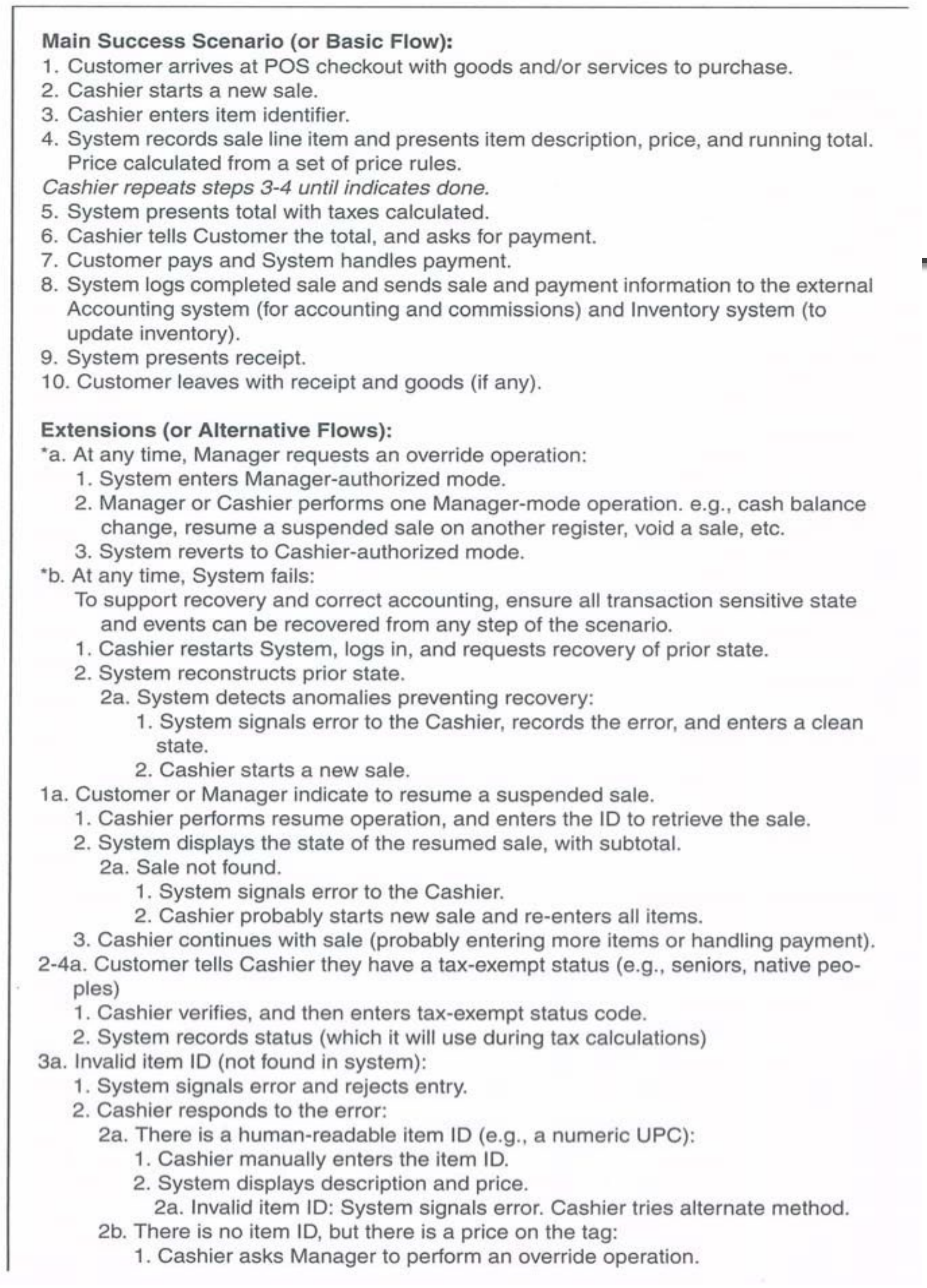
Use Case Section	Comment
Main Success Scenario	A record of the steps of a successful scenario, including interaction between actors, validation (by system), and state change by system. Steps are usually numbered.
Extensions	All other scenarios that may be branched to off the main success scenario; numbered according to main success scenario steps, and often this section is larger. May refer to another use case
Special Requirements	Non-functional requirements, quality attributes, constraints
Technology and Data Variations List	Any obvious technology or I/O constraints, data constraints
Misc	Anything else

Example: Process Sale, Fully Dressed Style



Use Case UC1: Process Sale

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.
Preconditions: Cashier is identified and authenticated.
Success Guarantee (or Postconditions): 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).

Extensions (or Alternative Flows):

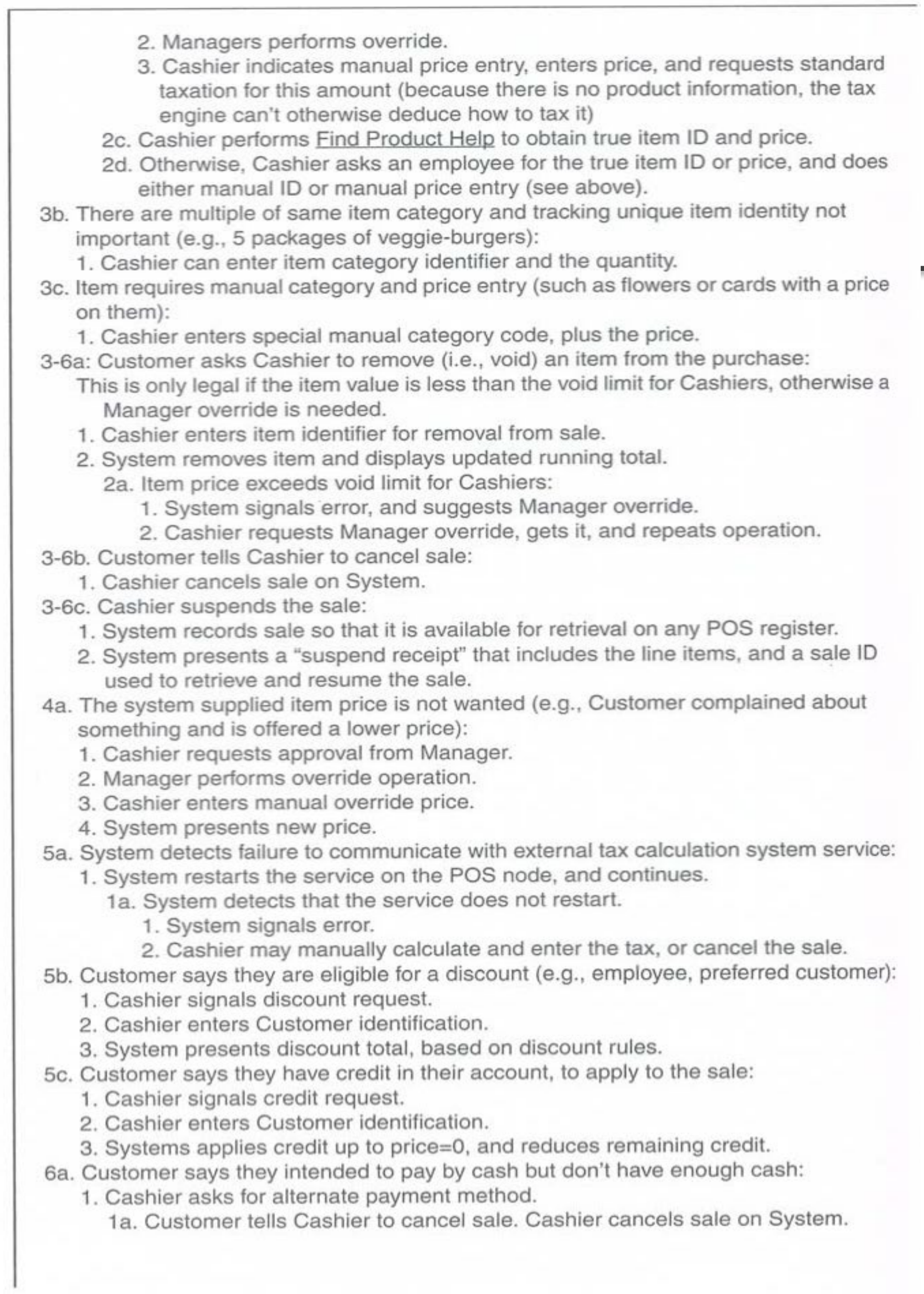
*a. At any time, Manager requests an override operation:
   1. System enters Manager-authorized mode.
   2. Manager or Cashier performs one Manager-mode operation. e.g., cash balance change, resume a suspended sale on another register, void a sale, etc.
   3. System reverts to Cashier-authorized mode.

*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.

1a. Customer or Manager indicate to resume a suspended sale.
   1. Cashier performs resume operation, and enters the ID to retrieve the sale.
   2. System displays the state of the resumed sale, with subtotal.
      2a. Sale not found.
         1. System signals error to the Cashier.
         2. Cashier probably starts new sale and re-enters all items.
   3. Cashier continues with sale (probably entering more items or handling payment).

2-4a. Customer tells Cashier they have a tax-exempt status (e.g., seniors, native peoples)
   1. Cashier verifies, and then enters tax-exempt status code.
   2. System records status (which it will use during tax calculations)

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).

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.

3c. Item requires manual category and price entry (such as flowers or cards with a price on them):
   1. Cashier enters special manual category code, plus the price.

3-6a: Customer asks Cashier to remove (i.e., void) an item from the purchase:
   This is only legal if the item value is less than the void limit for Cashiers, otherwise a Manager override is needed.
   1. Cashier enters item identifier for removal from sale.
   2. System removes item and displays updated running total.
      2a. Item price exceeds void limit for Cashiers:
         1. System signals error, and suggests Manager override.
         2. Cashier requests Manager override, gets it, and repeats operation.

3-6b. Customer tells Cashier to cancel sale:
   1. Cashier cancels sale on System.

3-6c. Cashier suspends the sale:
   1. System records sale so that it is available for retrieval on any POS register.
   2. System presents a "suspend receipt" that includes the line items, and a sale ID used to retrieve and resume the sale.

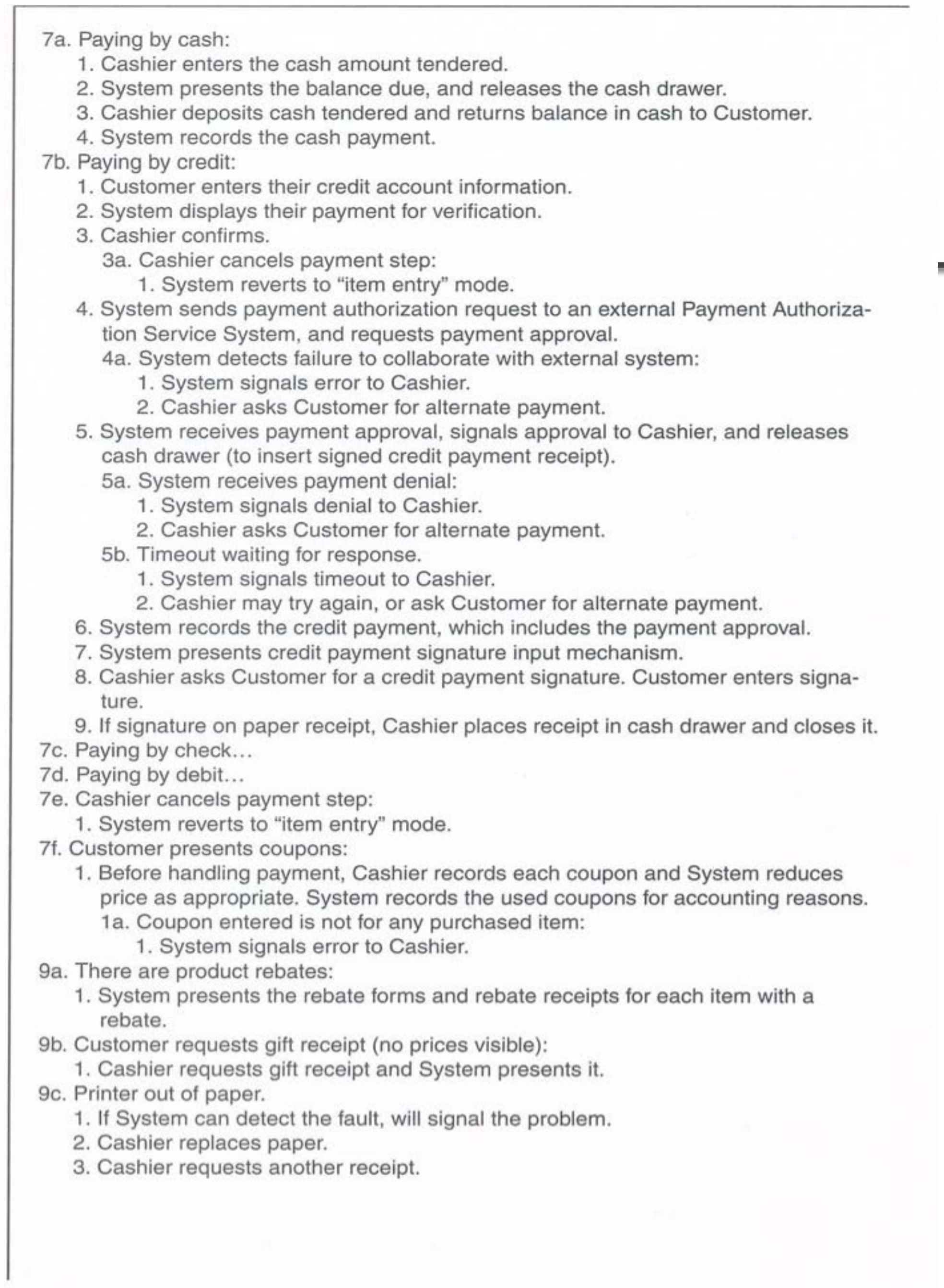
4a. The system supplied item price is not wanted (e.g., Customer complained about something and is offered a lower price):
   1. Cashier requests approval from Manager.
   2. Manager performs override operation.
   3. Cashier enters manual override price.
   4. System presents new price.

5a. System detects failure to communicate with external tax calculation system service:
   1. System restarts the service on the POS node, and continues.
      1a. System detects that the service does not restart.
         1. System signals error.
         2. Cashier may manually calculate and enter the tax, or cancel the sale.

5b. Customer says they are eligible for a discount (e.g., employee, preferred customer):
   1. Cashier signals discount request.
   2. Cashier enters Customer identification.
   3. System presents discount total, based on discount rules.

5c. Customer says they have credit in their account, to apply to the sale:
   1. Cashier signals credit request.
   2. Cashier enters Customer identification.
   3. Systems applies credit up to price=0, and reduces remaining credit.

6a. Customer says they intended to pay by cash but don't have enough cash:
   1. Cashier asks for alternate payment method.
      1a. Customer tells Cashier to cancel sale. Cashier cancels sale on System.



7a. Paying by cash:
  1. Cashier enters the cash amount tendered.
  2. System presents the balance due, and releases the cash drawer.
  3. Cashier deposits cash tendered and returns balance in cash to Customer.
  4. System records the cash payment.

7b. Paying by credit:
  1. Customer enters their credit account information.
  2. System displays their payment for verification.
  3. Cashier confirms.
    3a. Cashier cancels payment step:
      1. System reverts to "item entry" mode.
  4. System sends payment authorization request to an external Payment Authorization Service System, and requests payment approval.
    4a. System detects failure to collaborate with external system:
      1. System signals error to Cashier.
      2. Cashier asks Customer for alternate payment.
  5. System receives payment approval, signals approval to Cashier, and releases cash drawer (to insert signed credit payment receipt).
    5a. System receives payment denial:
      1. System signals denial to Cashier.
      2. Cashier asks Customer for alternate payment.
    5b. Timeout waiting for response.
      1. System signals timeout to Cashier.
      2. Cashier may try again, or ask Customer for alternate payment.
  6. System records the credit payment, which includes the payment approval.
  7. System presents credit payment signature input mechanism.
  8. Cashier asks Customer for a credit payment signature. Customer enters signature.
  9. If signature on paper receipt, Cashier places receipt in cash drawer and closes it.

7c. Paying by check...

7d. Paying by debit...

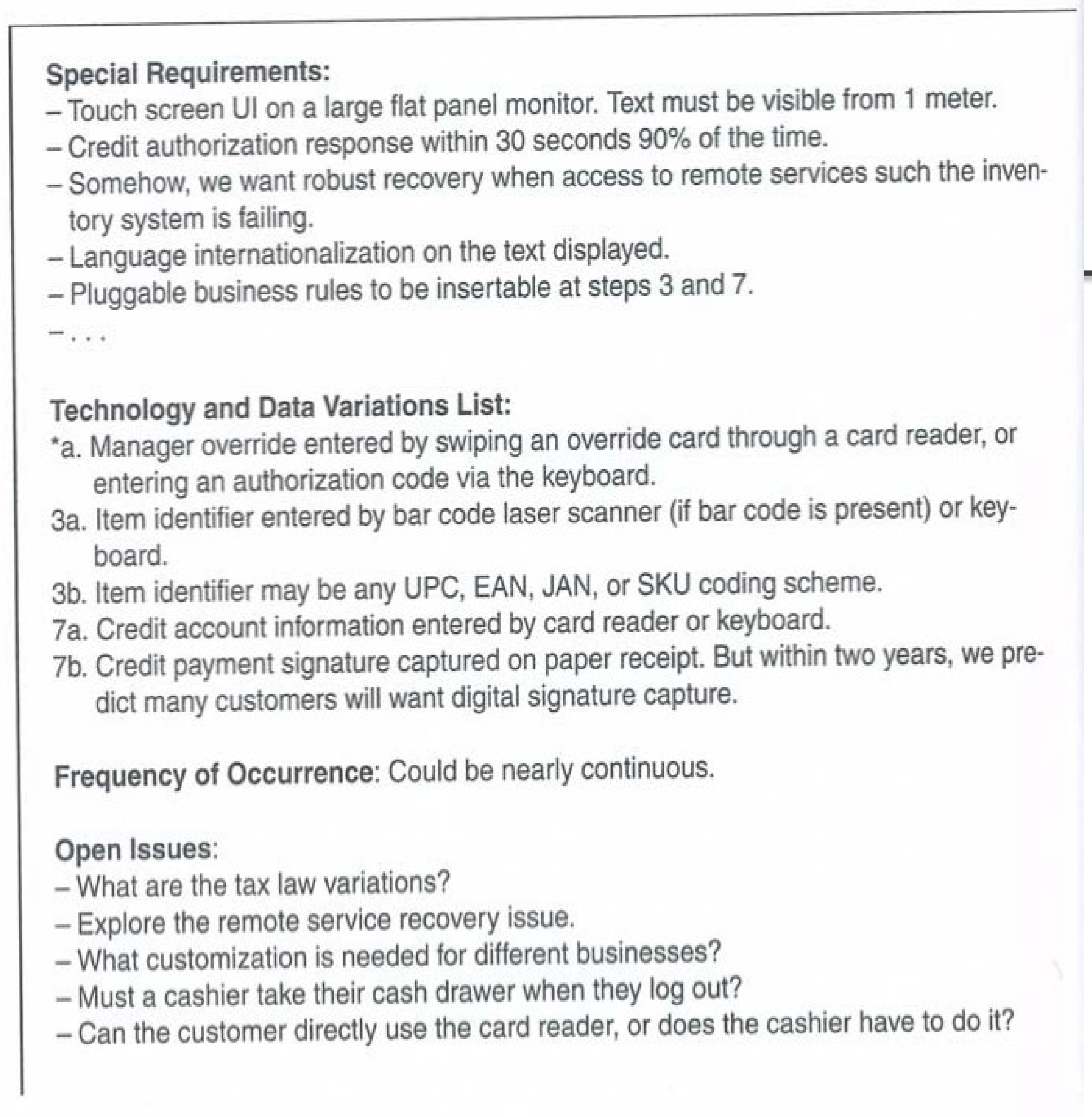
7e. Cashier cancels payment step:
  1. System reverts to "item entry" mode.

7f. Customer presents coupons:
  1. Before handling payment, Cashier records each coupon and System reduces price as appropriate. System records the used coupons for accounting reasons.
    1a. Coupon entered is not for any purchased item:
      1. System signals error to Cashier.

9a. There are product rebates:
  1. System presents the rebate forms and rebate receipts for each item with a rebate.

9b. Customer requests gift receipt (no prices visible):
  1. Cashier requests gift receipt and System presents it.

9c. Printer out of paper.
  1. If System can detect the fault, will signal the problem.
  2. Cashier replaces paper.
  3. Cashier requests another receipt.



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 insertable at steps 3 and 7.
– . . .

Technology and Data Variations List:

*a. 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?

Guideline: Write in an Essential UI-Free Style

✧ **Essential writing style** is to express user intentions and system responsibilities, rather than concrete actions.

- Concrete use cases are better avoided during early requirements analysis.
- For example: ***Manage Users*** use case

Essential Style

1. Administrator identities self.
2.System authenticates identity.
3. …

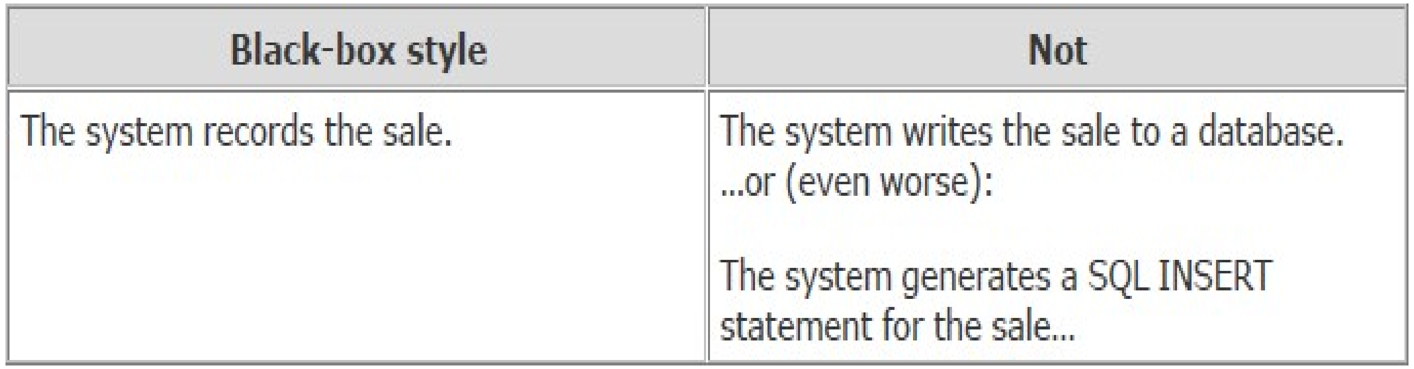
Concrete Style

1. Administrator enters ID and PW in dialog box.
2. System authenticates Administrator.
3.System displays the “edit user” window.
4. …

Guideline: Write Black-Box Use Cases

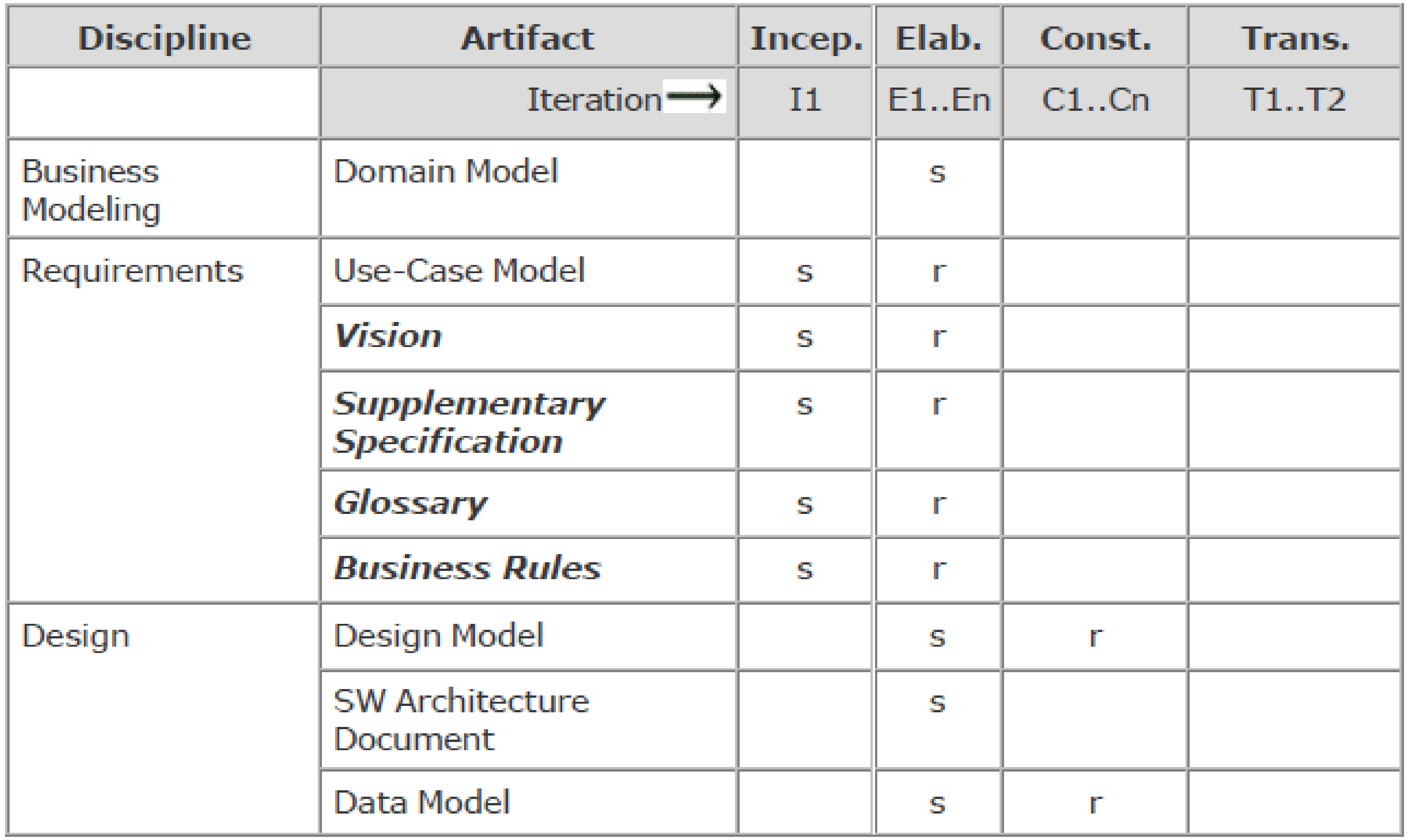
✧ Don’t describe **<u>the internal working</u>** of the system, its components or design.

- Define **what** the system does (analysis), rather than how it does it (design).



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...

Process: Evolutionary Requirements in Iterative Methods



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		
	Business Rules	s	r		
Design	Design Model		s	r	
	SW Architecture Document		s		
	Data Model		s	r	

Case Study: Use Cases in the NextGen POS

✧ Use cases are developed and refined iteratively.

✧ Use Cases of the NextGen POS at the inception phase

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

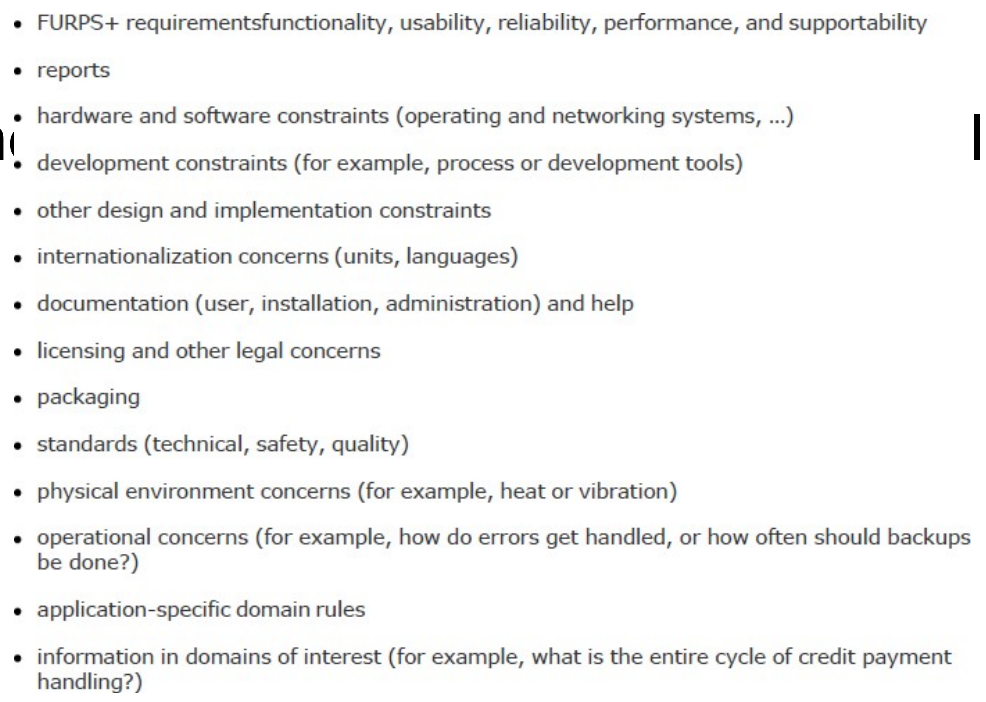
Other Requirements Artifacts

- ✧ **Supplementary Specification**
  - ▪ Captures and identifies **<u>other kinds of requirements</u>**, such as
    - reports, documentation, packaging, supportability, licensing, and so forth
- ✧ **Glossary**
  - ▪ Captures terms and definitions; a data dictionary
- ✧ **Vision**
  - ▪ Summarizes the “vision” of the project; an executive summary
- ✧ **Business Rules**
  - ▪ Capture long-living and spanning rules or policies (such as tax laws), that transcend one particular application

Supplementary Specification

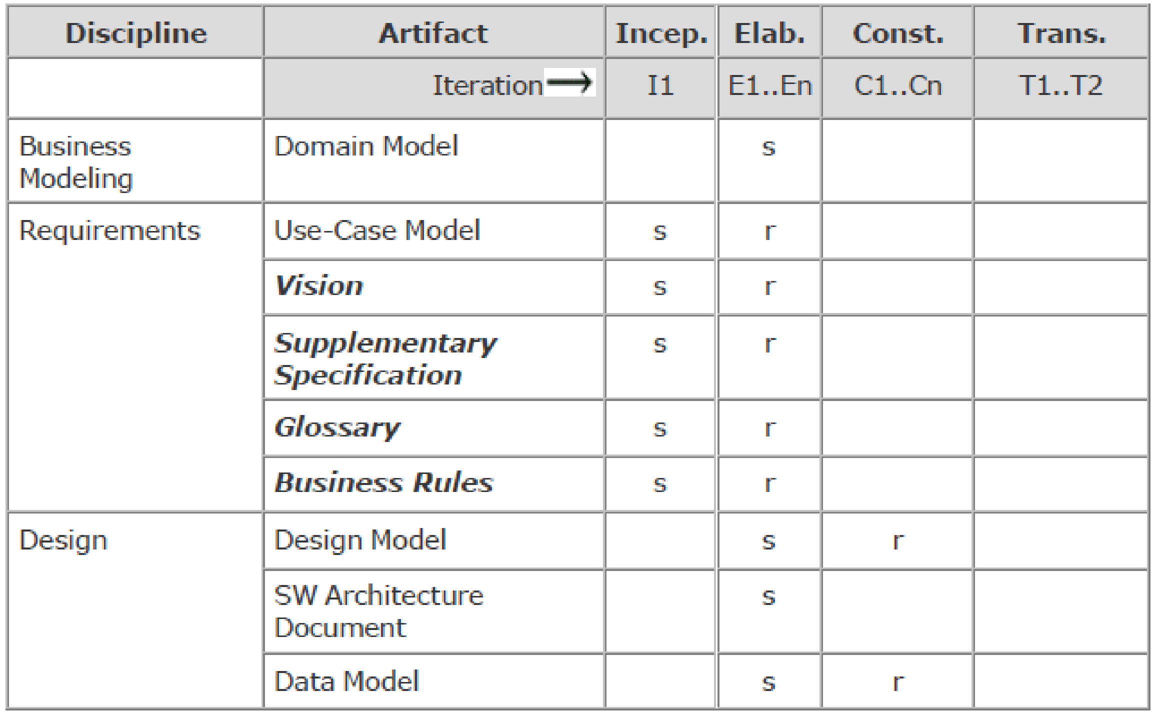
✧ Other requirements, information and constraints not easily captured in the use cases or Glossary, including system-wide "**URPS+**" **quality attributes**.

✧ Elements of th lude:



- FURPS+ requirementsfunctionality, usability, reliability, performance, and supportability
- reports
- hardware and software constraints (operating and networking systems, ...)
- development constraints (for example, process or development tools)
- other design and implementation constraints
- internationalization concerns (units, languages)
- documentation (user, installation, administration) and help
- licensing and other legal concerns
- packaging
- standards (technical, safety, quality)
- physical environment concerns (for example, heat or vibration)
- operational concerns (for example, how do errors get handled, or how often should backups be done?)
- application-specific domain rules
- information in domains of interest (for example, what is the entire cycle of credit payment handling?)

Process: Evolutionary Requirements in Iterative Methods



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		
	Business Rules	s	r		
Design	Design Model		s	r	
	SW Architecture Document		s		
	Data Model		s	r	