So Far …

Part 1: OOAD Intro

- SE
- OOAD
- UML
- Iterative, Evolutionary, and Agile
- Case Studies

Part 2: Inception

- Inception is Not the Requirements Phase
- Evolutionary Requirements
- Use Cases
- Other Requirements

Part 3: Elaboration—Iteration 1

- Iteration 1—Basics
- Domain Models
- ….

Use Cases

Abdulkareem Alali

ACK Dale Haverstock

Based on Larman’s Applying UML and Patterns Book, 3d

Use Case Format

There isn't a single best format for use cases

There is a two-column variation

Use Case Writing Guidelines —Essential style?

Write use cases in an **Essential Style**

- keep the UI details out and focus on actor intent

Write as tersely as possible (drop The), avoid unnecessary verbiage (noise)

Essential Style
Assume that *Manage Users* use case requires identification and authentication:
1. Administrator identifies self.
2. System authenticates identity.
3. …

The design solution to these intentions and responsibilities is wide open: biometric readers, GUIs, and so forth.

Concrete Style—Avoid During Early Requirements Work
User interface decisions are embedded in the use case text.

1. Administrator enters ID and password in dialog box (see Picture 3).
2. System authenticates Administrator.
3. System displays the "edit users" window (see Picture 4).

…

Use Case Writing Guidelines —Black-box Use Cases

Write **Black-Box Use Cases** interactions between actors and the system

Do not attempt to describe the internal workings of the system

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...

Use Case Writing Guidelines —Actor/Goal Perspective

"**analysis**" versus "**design**" is sometimes summarized as "**what**" versus "**how**"

Take an actor and **actor-goal** perspective:

An observable result of value to a particular actor is what is desired.

"What do you do?“ vs.

"What are your goals whose results have measurable value?"

Use Case Writing Guidelines—Take an Actor/-Goal Perspective

In general, define one use case for **each user goal**

The use case should have a name similar to the user goal

- E.g. If the goal is "process a sale",
- then a good use case name would be **Process Sale**

Use Case Writing Guidelines—Take an Actor/-Goal Perspective

The name of a use cases should begin with a **verb**

Collapse **CRUD** (create, retrieve, update, delete)

E.g., the goals “edit user,” “delete user,” and so forth are all satisfied by the **Manage Users** use case.

Finding Use Cases

1. Identify the system **boundary**

   It could be just a software application, or hardware and software as a unit, or with a person using it, or an entire organization

2. Identify the **primary actors**
3. Identify the **goals for each primary actor**
4. Name the **use cases according to their goal** and define them

UML use case diagrams can be used to show the relationships between actors and use cases

More on Actors and Goals

Goals can reveal the actors, or vice versa find users

Brainstorm with the primary actors first, this will help set-up further investigation

Primary actors are the most obvious and the easiest relatively

More on Actors and Goals

Questions to help identify other actors and goals:

- Who starts and stops the system?
- Who does system administration?
- Who does user and security management?
- Who gets notified when there are errors or failures?
- Who evaluates system activity or performance?

More on Actors and Goals

Questions to help identify other actors and goals:

- Who evaluates logs? Are they remotely retrieved?
- Are there any external software or automated systems that call upon services of the system?
- Is there a monitoring process that restarts the system if it fails?
- How are software updates handled?

Actors and Goals Table

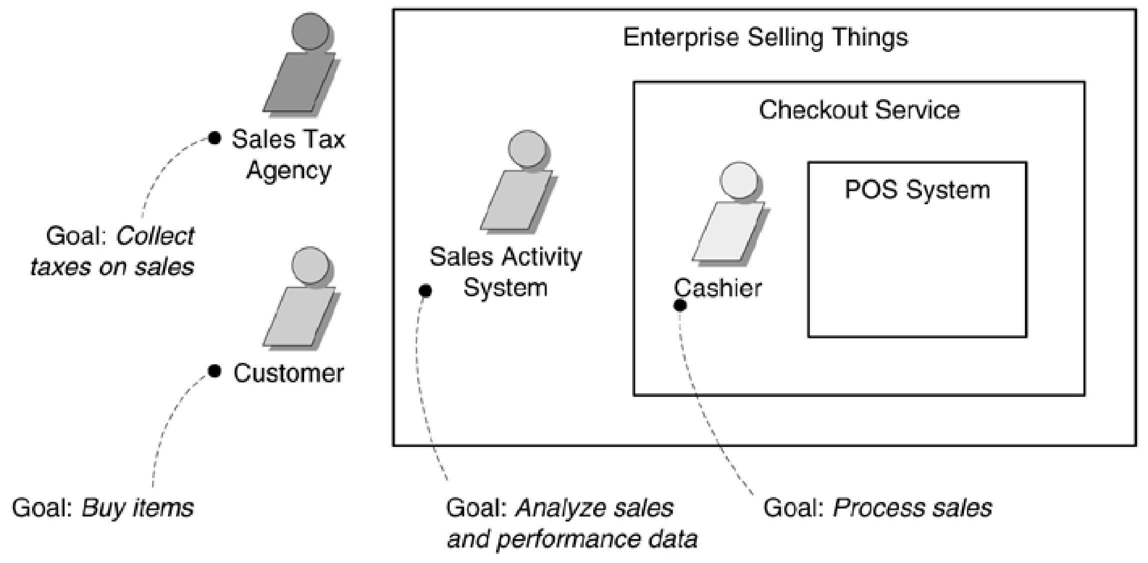
Actor	Goal		Actor	Goal
Cashier	process sales process rentals handle returns cash in cash out …		System Administrator	add users modify users delete users manage security manage system tables …
Manager	start up shut down …		Sales Activity System	analyze sales and performance data
…	…		…	…

POS –Is Customer the Primary Actor?

System is serving the goals of the **Cashier**, thus the cashier is the primary actor

From a higher-level business perspective, and a different system boundary, the **Cashier** could be viewed as part of the system (e.g.?) and the **Customer** as the primary actor

Given the POS system the Customer wouldn't be able to use it



Identifying Useful/Good Size Use Cases

What constitutes a useful use case?

- Negotiate a Supplier Contract
    - Or Research the User Experience
- Handle Returns
- Log In
- Move Piece on Game Board

A goal of measurable value to be achieved.

Identifying Useful/Good Size Use Cases

Some tests:

1. **Boss Test**
2. **EBP Test**
3. **Size Test**

The Boss Test

The boss asks, "What have you been doing?"

You answer, "logging in".

The boss is probably not happy. "logging in" fails the boss test

Login may be a use case at some low goal level; however, it is not at the desirable level of focus for requirements analysis

User authentication may be important and difficult though, may not pass? Okta?

The EBP Test

An **elementary business processes** (EBP) is 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

A transaction

e.g.

- Approve Credit
- Place Order
- Create a wish list

The EBP Test

Literally? Two people? person has to walk around? Probably not

“delete a line item” or “print the document.”

It is a task done during a single session. It is probably between a few minutes and an hour in length

Adding observable or measurable business value, a resolution, data are in a stable and consistent state

The Size Test

A use case is very seldom a single action or step

Use case main success scenario is probably five or ten steps

Fully dressed format will often require 3–10 pages of text

e.g. Enter an Item ID

- You can see a hint of the error by its small size
- Imagine the length of its fully dressed text, it would be extremely short

Example: Applying the Tests

Negotiate a Supplier Contract or Research the User Experience

- Much broader and longer than an EBP

Handle Returns

- OK with the boss. Seems like an EBP. Size is good.

Log In

- Boss not happy if this is all you do all day!

Move Piece on Game Board

- Single step—fails the size test

Exceptions

User-goal level vs **Sub-function level**

Common **sub-tasks** might be best regarded as separate use cases

- E.g., a subtask or extension such as "**Paying by credit**" may be repeated in several base use cases
- Separate into its own use case, even fails EBP and size tests
- link it to several base use cases, to **avoid duplication** of text

Again, **Authenticate User** may fail the Boss test, but it may be complex enough to warrant careful analysis

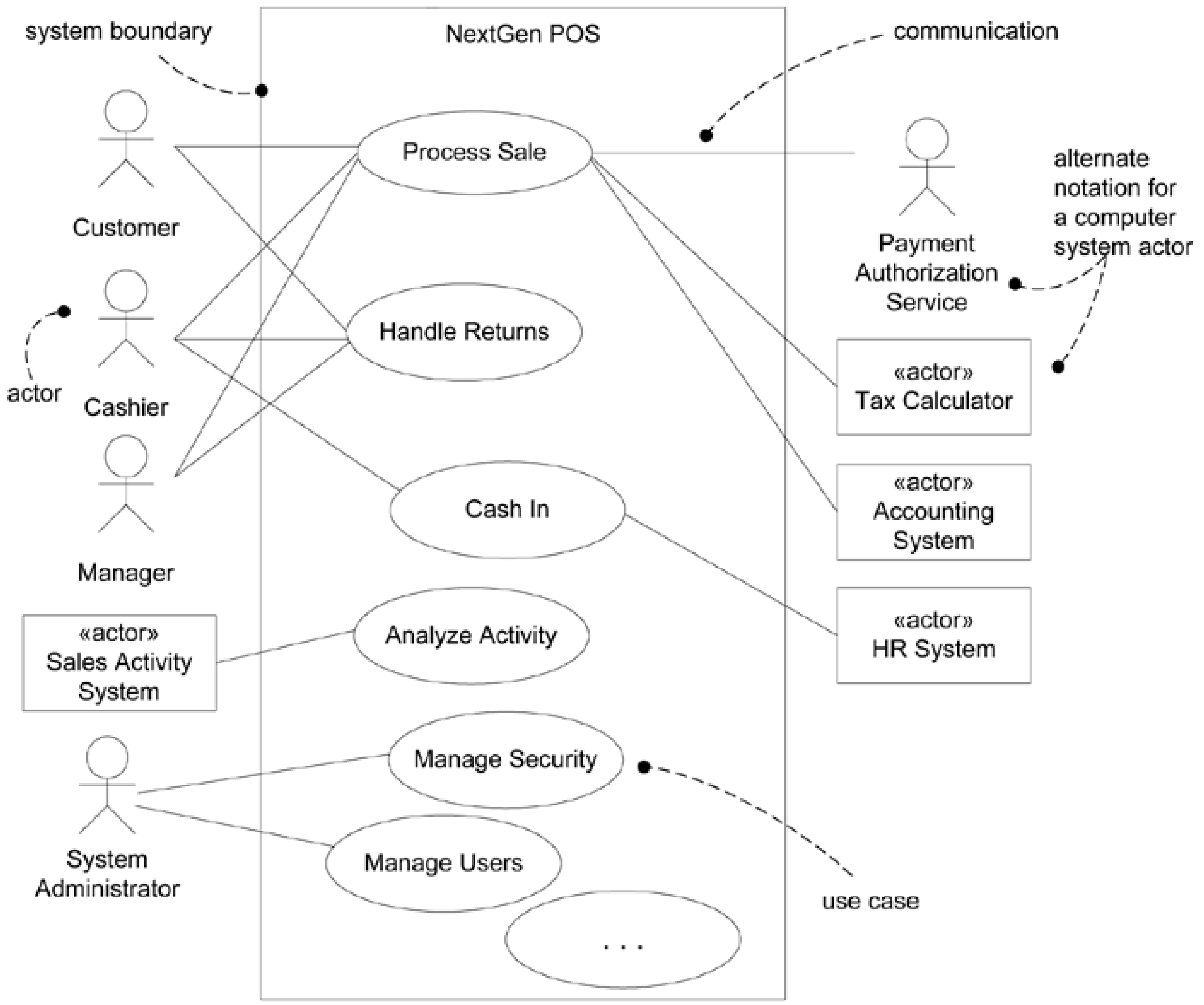
Applying UML —Use Case Diagrams

UML use case diagrams show the relationships between actors and use cases and between use cases

Use cases are text documents

A focus on use case diagrams indicates a misunderstanding of requirements elicitation

POS Partial Use Case Diagram



Applying UML —Activity Diagrams

Visualize workflows and business processes

Use cases involve process and workflow analysis

A useful alternative or adjunct to writing the use case text, but **why not a sequence diagram?!**

Business use cases that describe complex workflows involving many parties and concurrent actions

Other Approaches vs. Use Cases —High-Level System Feature Lists are Acceptable

Feature lists used to be used to specify requirements

- The system shall accept entry of bar codes
- The system shall log credit payments to the accounts receivable system.
- …

There is some software where a use case approach is not useful. e.g. Creating software Language or a DBMS

ID	Feature
FEAT1.9	The system shall accept entry of item identifiers.
…	…
FEAT2.4	The system shall log credit payments to the accounts receivable system.

VS

ID	Feature
FEAT1	Sales capture
FEAT2	Payment authorization (credit, debit, check)
FEAT3	System administration for users, security, code and constants tables, and so on
…	…

Detailed Feature Lists Appropriate Rather than Use Cases?

Sometimes use cases do not really fit

Some applications cry out for a feature-driven viewpoint

- Application servers, database products

Evolved in terms of features ("We need Web Services support in the next release")

Use cases are not a natural fit for these applications or the way they need to evolve in terms of market forces

Two Approaches To Development [McLaughlin *et. al.*] [.]

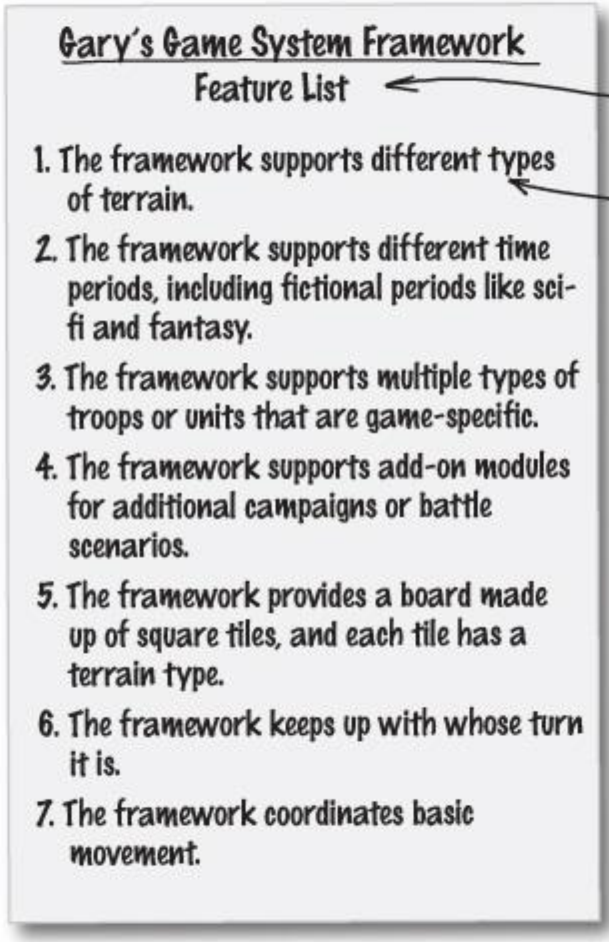
Feature Driven Development

Use Case Driven Development

Feature Driven Development

- You work on a single feature at a time,
- then iterate,
- knocking off features one at a time until you’ve finished up the functionality of an application

Feature Driven Development .

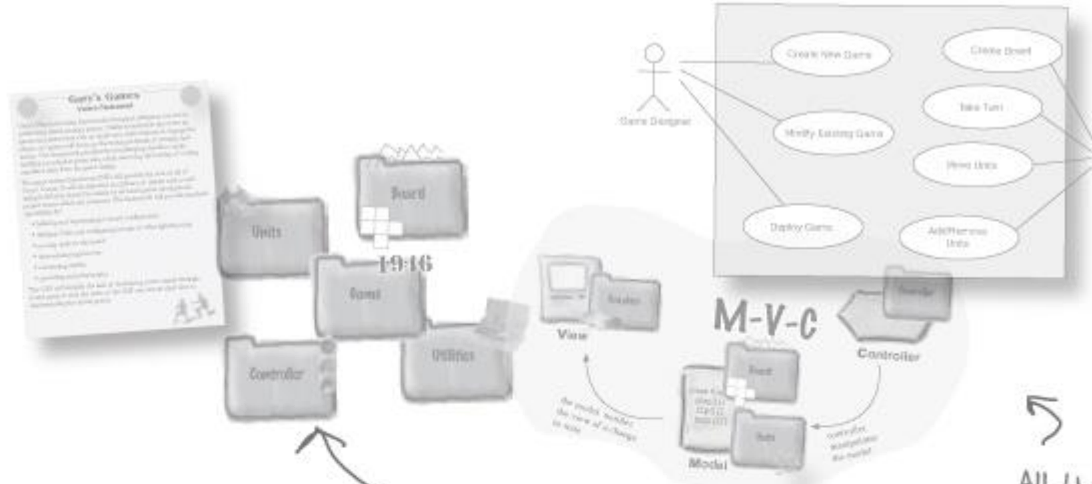


Gary's Game System Framework
Feature List

1. The framework supports different types of terrain.
2. The framework supports different time periods, including fictional periods like sci-fi and fantasy.
3. The framework supports multiple types of troops or units that are game-specific.
4. The framework supports add-on modules for additional campaigns or battle scenarios.
5. The framework provides a board made up of square tiles, and each tile has a terrain type.
6. The framework keeps up with whose turn it is.
7. The framework coordinates basic movement.

With feature driven development, you pick a single feature, and the focus is on the feature list of your app.

So we might take feature #1, and work on the Terrain class, as well as the Tile class, to support different types of terrain.

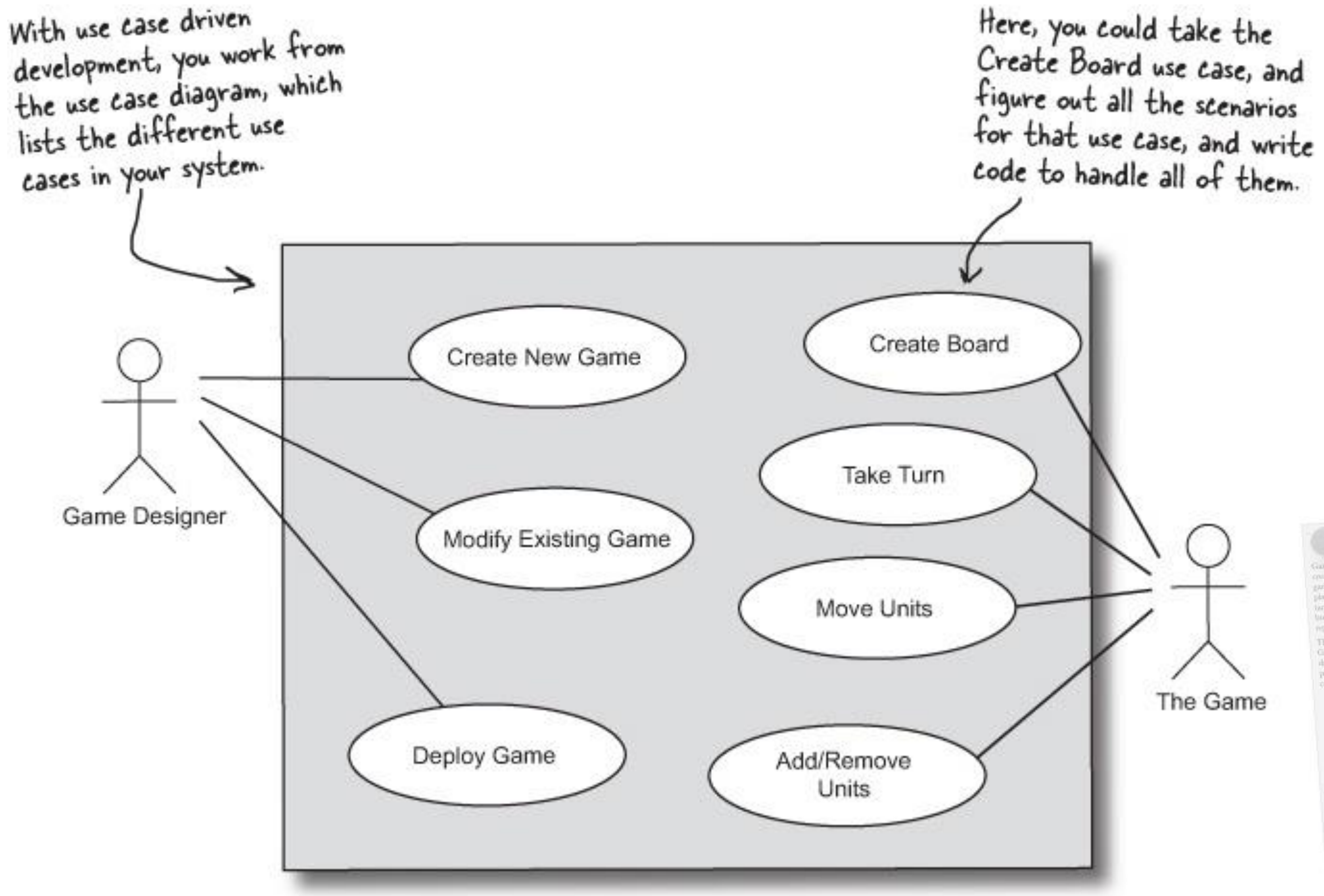


All these other plans and diagrams are used, but your feature list is the focus.

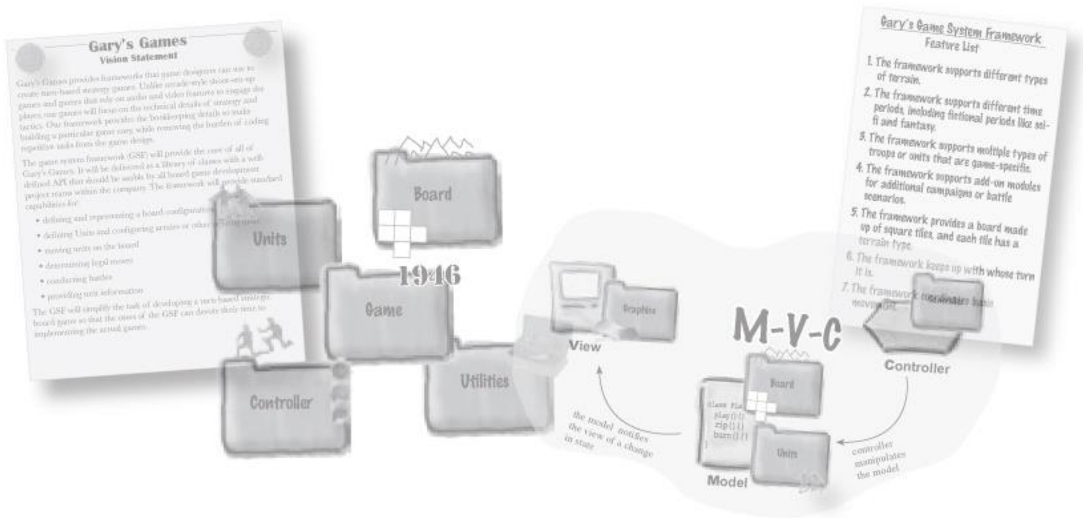
Use Case Driven Development

- You work on completing a single scenario through a use case
- Then you take another scenario and work through it, until all of the use case's scenarios are complete
- Then you iterate to the next use case, until all your use cases are working.

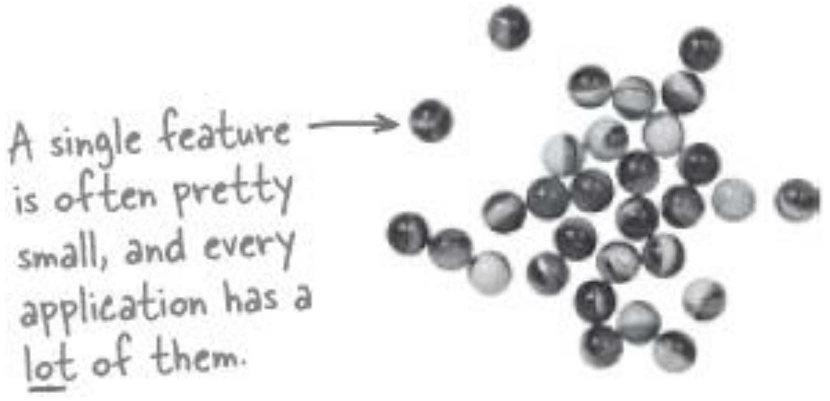
Use Case Driven Development [.]


With use case driven development, you work from the use case diagram, which lists the different use cases in your system.
Here, you could take the Create Board use case, and figure out all the scenarios for that use case, and write code to handle all of them.
Create New Game
Create Board
Take Turn
Modify Existing Game
Move Units
Deploy Game
Add/Remove Units
Game Designer
The Game

—Bit Forced on some types of Games!



Difference Between Feature Driven And Use Case Driven Development .


A single feature is often pretty small, and every application has a lot of them.

You'll be working on pretty major chunks of code at a time, since a single scenario often involves a lot of functionality.

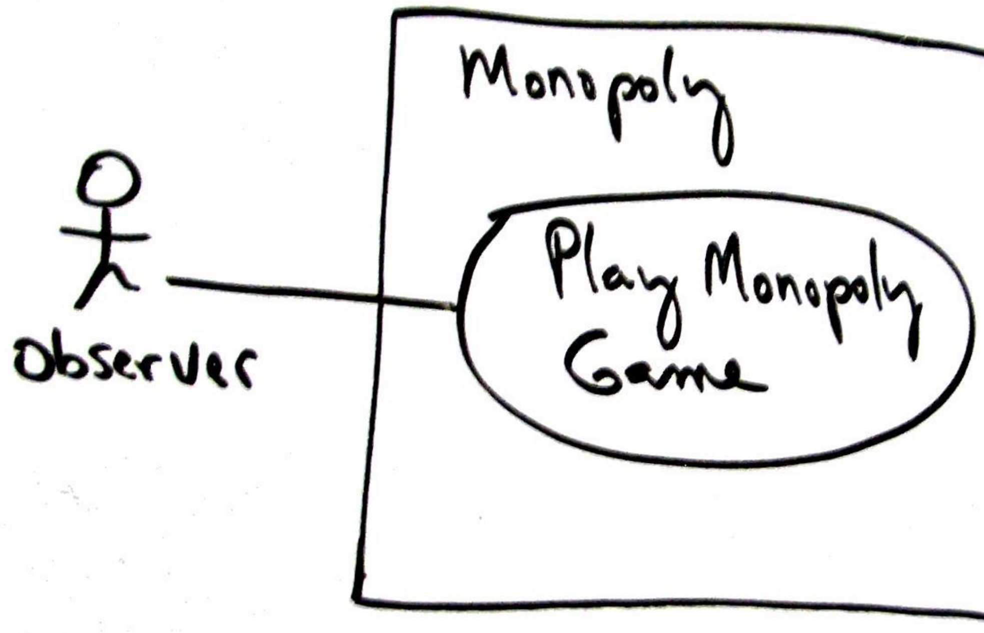
Difference Between Feature Driven And Use Case Driven Development

Feature driven development is more granular	Use case driven development is more “big picture”
Works well when you have a lot of different features that don’t interconnect a whole lot	Works well when your app has lots of processes and scenarios rather than individual pieces of functionality
Allows you to show the customer working code faster	Allows you to show the customer bigger pieces of functionality at each stage of development.
Is very functionality-driven. You’re not going to forget about any features using feature driven development	Is very user-centric. You’ll code for all the different ways a user can use your system with use case driven development

Name That Approach!

	Use Case Driven	Feature Driven
This approach deals with really small pieces of your application at a time.		✓
This approach lets you focus on just a part of your application at a time.	✓	✓
This approach is all about a complete process in your application.	✓	
Using this approach, you can always test to see if you've completed the part of the application you're working on.	✓	✓
When you use this approach, your focus is on a diagram, not a list.	✓	

Monopoly Use Case Diagram



Monopoly Use Case Text:

- **Scope**: Monopoly Application
- **Level**: User Goal
- **Primary Actor**: Observer
- **Stakeholders**:
  Observer: easily observe game simulation output
- **Main Scenario**:
  1. Observer requests new simulation, enters num players
  2. Observer starts play.
  3. System displays game trace after each play
  4. *Repeat 3. until game over or Observer cancels*

Monopoly Use Case Text:

- Extensions:
  - **a*: At any time, system fails

    (System logs each move)

    1. Observer restarts system
    2. System detects failure and reconstructs correct state, continues
    3. Observer chooses to continue
- Special Requirements:
  - Provide graphical and text trace modes

Monopoly Game

Domain rules or legal rules or business rules part of the Supplementary Specification (SS) more than scenarios

Trying to capture all the game rules in the use case format is unnatural

Use Cases Under The Process

Use Cases and Iterative Methods

The UP encourages **Use Case driven development**

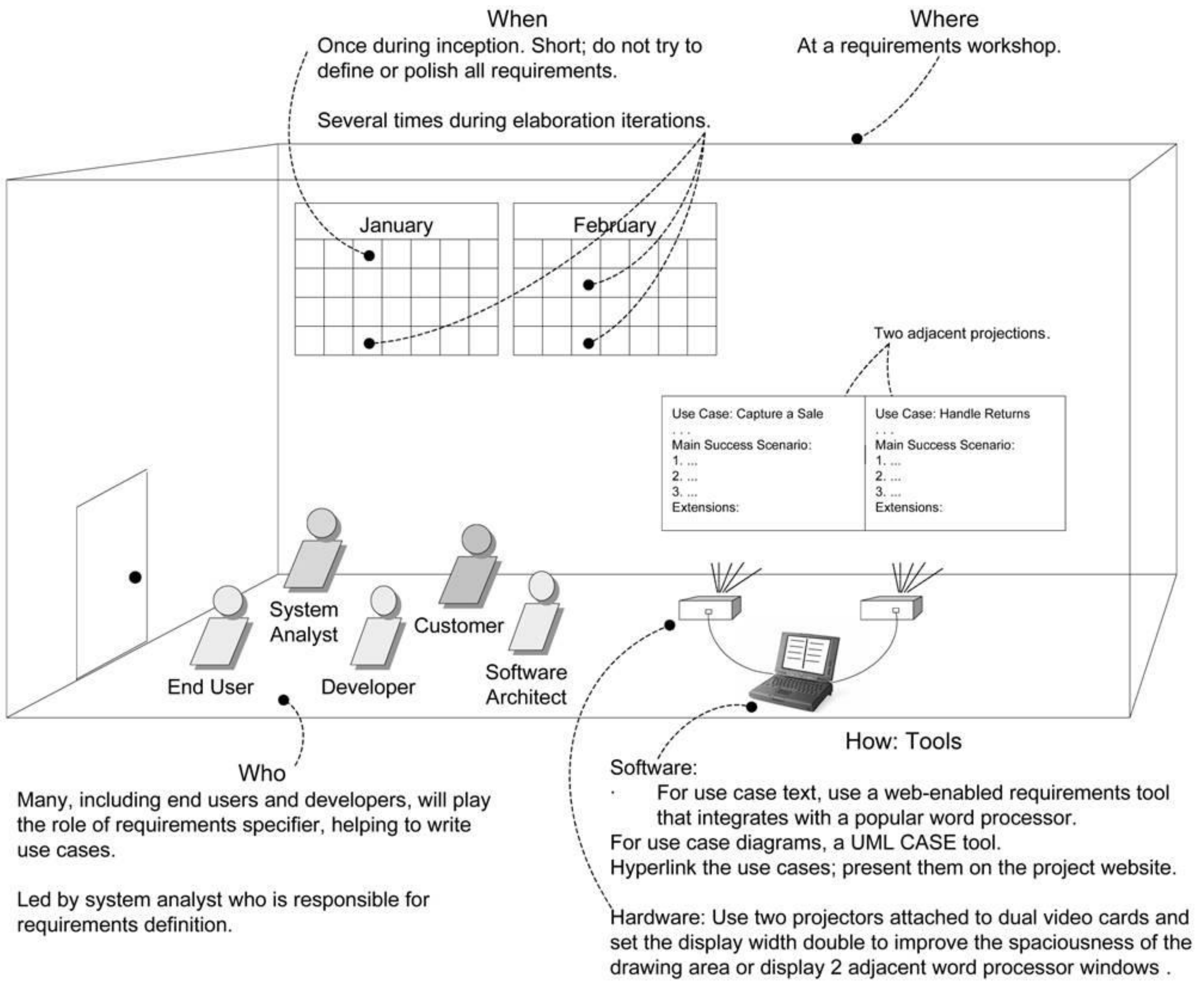
Most of the **use cases are written incrementally** in the **iterations** of the elaboration phase

By the end of elaboration, a large body of detailed use cases have been written which provides a realistic basis for estimation through the remainder of the project

Table 6.1. Sample requirements effort across the early iterations; this is not a recipe. [Larman]

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

Process And Setting Context For Writing Use Cases


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 adjacent word processor windows .

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

Table 6.2. Sample UP artifacts and timing. s - start; r - refine

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

How to Write Use Cases in Inception?

- 2-day requirements workshop during the early POS investigation
- The earlier part of the day is identifying goals, stakeholders, and scope
- An actor-goal-use case table is written and displayed with the computer projector
- A use case diagram is started
- After a few hours, perhaps 20 use cases are identified by name for POS
- Most of the interesting, complex, or risky use cases are written in brief format

How to Write Use Cases in Inception?

- The team starts to form a high-level picture of the system's functionality
- 10% to 20% of the use cases that represent core complex functions, require building the core architecture, or risky in some dimension are rewritten in a fully dressed format
- The team investigates a little deeper to better comprehend the magnitude, complexities, and hidden demons of the project through deep investigation of a small sample of influential use cases
    - Process Sale and Handle Returns.

Case Study: Use Cases in the POS NextGen Inception Phase

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

How to Write Use Cases in Elaboration?

- Time-boxed iterations (e.g. four iterations) in risky, high-value, or architecturally significant parts of the system are incrementally built, and the "**majority**" of requirements identified and clarified.
- Feedback from the concrete steps of programming influences and informs the team's understanding of the requirements, which are iteratively and adaptively refined
- Perhaps there is a two-day requirements workshop in each iteration—four workshops.

How to Write Use Cases in Elaboration?

- 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
- 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.

How to Write Use Cases in Elaboration?

- 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 POS NextGen team should not only have a better definition of the use cases, but some quality executable software

How to Write Use Cases in Construction?

The construction phase is composed of time-boxed iterations (for example, 20 iterations of two weeks each)

Focus on completing the system, once the risky and core unstable issues have settled down in elaboration

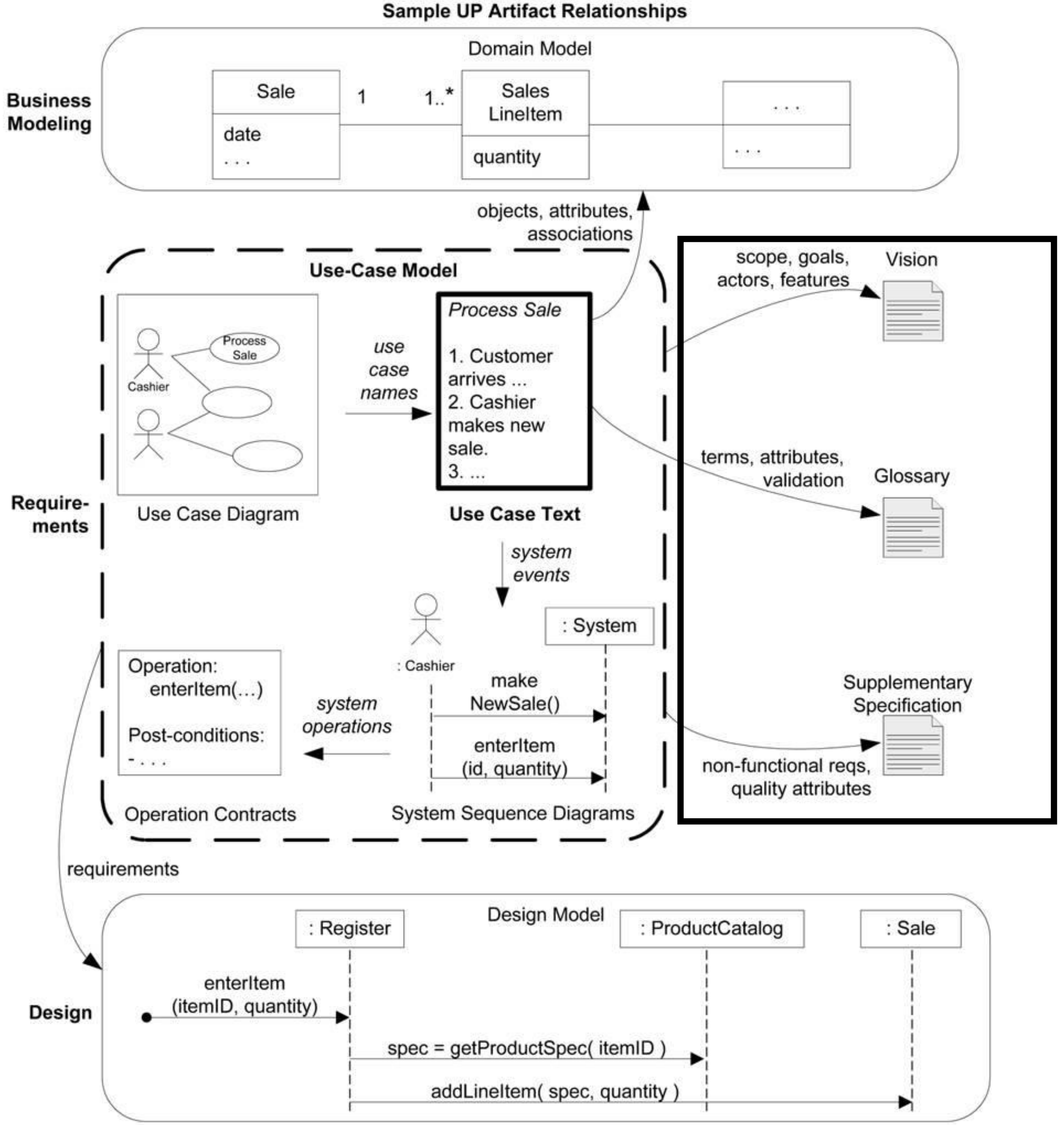
Some minor use case writing and perhaps requirements workshops, but much less so than in elaboration

Other Requirements

Abdulkareem Alali

ACK Dale Haverstock

Based on Larman’s Applying UML and Patterns Book, 3d


Sample UP Artifact Relationships
Business Modeling
Domain Model
Sale
date
. . .
1
1..*
Sales LineItem
quantity
. . .
. . .
objects, attributes, associations
Use-Case Model
Process Sale
Cashier
Use Case Diagram
use case names
Process Sale
1. Customer arrives ...
2. Cashier makes new sale.
3. ...
Use Case Text
scope, goals, actors, features
Vision
terms, attributes, validation
Glossary
Requirements
system events
: System
: Cashier
make NewSale()
enterItem (id, quantity)
Operation: enterItem(…)
Post-conditions: - . . .
system operations
Supplementary Specification
non-functional reqs, quality attributes
Operation Contracts
System Sequence Diagrams
requirements
Design Model
: Register
: ProductCatalog
: Sale
enterItem (itemID, quantity)
Design
spec = getProductSpec(itemID)
addLineItem(spec, quantity)

Other Requirements

Our focus is **OOAD** and not other items which are necessary for the management of the process

We will reiterate there are other requirements beyond use cases and take a look at some in a small way

Other Requirement Artifacts —We've mentioned

Supplementary Specification

—Keeps track of other requirements, the **URPS+, F**?

Glossary

—Keep track of terminology and data dictionary

Vision

—An executive summary, a terse description of the big ideas

Business Rules (or Domain Rules)

—Keep track of long-living and spanning rules or policies.
e.g.tax laws.

Should We Analyze These Other Requirements Thoroughly During Inception?

No. UP is an iterative/evolutionary method, production-quality programming and testing should happen very early, long before most requirements have been fully analyzed or recorded

- Requirements will evolve due to feedback from early programming and tests
- It is useful to have a high-level "top ten“, big points of main features
- Investigating the most important non-functional requirements
- Can have a significant impact on architectural choices

Reliable Specifications?

Goal: Build software that passes the acceptance tests defined by the users, software that meets their true goals.

True goals of the clients are often not discovered until they are evaluating and working with the software

Artifacts will evolve as development progresses

FURPS+

Functional (features, capabilities, security)

Usability (human factors, help, documents)

Reliability (failures, recovery, predictable)

Performance (response, throughput)

Supportability (maintainability, configuration)

+ constraints (next slide)

Constraints

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

NextGen POS (Partial) —Supplementary Specification

Introduction

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

Logging and **Error Handling**

- Log all errors to persistent storage

Security

- All usage requires user authentication

NextGen POS (Partial) —Supplementary Specification

Usability (Human Factors)

- Customer will be able to see a large-monitor display of the **POS**
    - Text should be easily visible from 1 meter
    - Avoid colors associated with common forms of color blindness
- Speed, ease, and error-free processing, buyer wishes to leave quickly
- Cashier is often looking at the customer or items, not the computer display signals/warnings should be conveyed with sound or light rather only graphics

NextGen POS (Partial) —Supplementary Specification

Reliability

- Recoverability
    - In case of failure to external services, solve locally
- Performance
    - Buyers want to complete sales processing very quickly. One bottleneck is external payment authorization. Our goal: Authorization in less than 1 minute, 90% of the time

NextGen POS (Partial) —Supplementary Specification

Supportability

- Adaptability
  - “Pluggable" business rules, Different customers of the POS have unique business rule while processing a sale

- Configurability
  - Different customers desire varying network configurations for their POS systems, such as thick versus thin clients (most resources installed locally vs. distributed over a network)
  - Ability to modify these configurations, to reflect their changing business and performance needs, Flexibility!

NextGen POS (Partial) —Supplementary Specification

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

- Maximize use of free Java open source (e.g. JLog logging framework)

NextGen POS (Partial) —Supplementary Specification

Interfaces

- Hardware
  - Touch screen monitor
  - Barcode laser scanner
  - Receipt printer
  - Credit/debit card reader
  - Signature reader

- Software
  - Plug in varying systems and thus varying interfaces (tax calculator, accounting, inventory, …)

NextGen POS (Partial) —Supplementary Specification

Application-Specific (e.g. Process Sales) Domain Rules

ID	Rule	Changeability	Source
RULE1	Purchaser discount rules. Examples: Employee—20% off. Preferred Customer—10% off. Senior—15% off.	High. Each retailer uses different rules.	Retailer policy.
RULE2	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 from 10am to 3pm today. Tofu 50% off from 9am-10am today.	High. Each retailer uses different rules, and they may change daily or hourly.	Retailer policy.
RULE3	Product (line item level) discount rules. Examples: 10% off tractors this week. Buy 2 veggie burgers, get 1 free.	High. Each retailer uses different rules, and they may change daily or hourly.	Retailer policy.

NextGen POS (Partial) —Supplementary Specification

Legal Issues

- Software licensing
- All tax rules must, by law, be applied during sales

Domains of Interest

- Pricing (Organizations maintain the original price for accounting and tax reasons)
- Credit and Debit Payment Handling
- Sales Tax (delegating tax calculations to third-party calculator software)
- Item Identifiers: UPCs, EANs, SKUs, Bar Codes … POS needs to support various item identifier schemes

Supplementary Specification Comments

Supplementary Specification **captures other requirements**, information, and constraints not easily captured in the use cases or Glossary

Some non-functional requirements may be initially written with a use case but as they often transcend use cases they are better moved to the Supplementary Specification

NextGen POS (Partial) —Vision

Introduction

We envision a next generation fault-tolerant point-of-sale (POS) application, NextGen POS, with the flexibility to support varying customer business rules, multiple terminal and user interface mechanisms, and integration with multiple third-party supporting systems.

NextGen POS (Partial) —Vision

Positioning

- Business Opportunity (adaptability, scale, integration, new devices PDAs problems.)
- Problem Statement (inflexible leads to shortage in sales, effects all stakeholders)
- Product Position Statement (outstanding features)
- Alternatives and Competition

NextGen POS (Partial) —Vision

Summary of System Features

- Sales capture
- Payment authorization (credit, debit, check)
- System administration for users, security, code and constants tables, and so forth.
- Automatic offline sales processing when external components fail
- Real-time transactions, based on industry standards, with third-party systems, including inventory, accounting, human resources, tax calculators, and payment authorization services
- Definition and execution of customized "pluggable" business rules at fixed, common points in the processing scenarios

Vision Commentary

It is useful to be to say to a newcomer "Welcome! Please go read the 7-page Vision at the project website.“

Establish a common vision of the project

In the Vision, system features briefly summarize functional requirements that are detailed in the use cases

Avoid their duplication or near-duplication in both the Vision and Supplementary Specification (SS)

NextGen POS (Partial) —Glossary

Example Glossary terms (that are defined), Data Dictionary

Term	Definition and Information	Format	Validation Rules	Aliases
item	A product or service for sale			
payment authorization	Validation by an external payment authorization service that they will make or guarantee the payment to the seller.			
payment authorization request	A composite of elements electronically sent to an authorization service, usually as a char array. Elements include: store ID, customer account number, amount, and timestamp.			
UPC	Numeric code that identifies a product. Usually symbolized with a bar code placed on products. See **www.uc-council.org** for details of format and validation.	12-digit code of several subparts.	Digit 12 is a check digit.	Universal Product Code
...	...			

Glossary Comments

Guideline: Start the Glossary early.

Composite Terms e.g. “product price”

In the UP, the Glossary also functions as the **data dictionary**, a document that records data about the data-that is, metadata

POS NextGen —Business Rules (Domain Rules)

General Business Rules, examples:

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 government 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

Domain Rules Comments

Domain rules dictate how a domain or business may operate

Common domain rules come from:

Government laws,

Company policies,

and Physical laws