

## Agile Use Cases (Driving Development)



[www.netobjectives.com](http://www.netobjectives.com)  
[Doug\\_shimp@netobjectives.com](mailto:Doug_shimp@netobjectives.com)

## My Goals



Enlighten



Teach



Inspire

## Purpose of Seminar

- Describe Fundamentals of Agility
- Show how Use Cases are an “obvious” way to drive Agile Development
- Describe Basics of Use Case development
- Straight-ahead, no nuance, sacrifice accuracy for truth...

## Agenda


- Agility
  - Software Development
    - Purpose
    - Philosophies
  - Doing the Work in an Agile Project
    - Backlog
    - Stories
    - Tasks
  - Concept of Driving with Use Cases
- Use Cases Driven Agile Development
- Use Case Development
- Summary

## The Purpose of Software Development is...

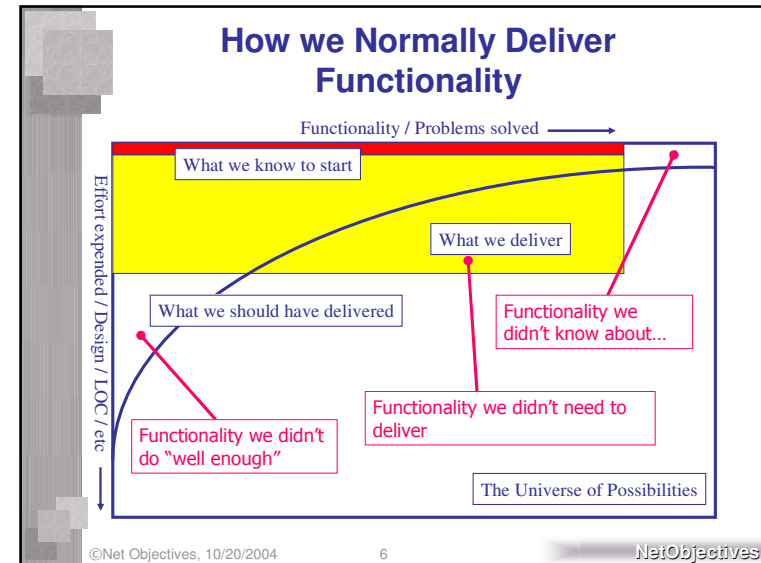
To provide a suitable solution for users...  
That consists of quality code...  
And doesn't cost too much

**Where Use Cases Live**

**Suitable = Useful + Usable**



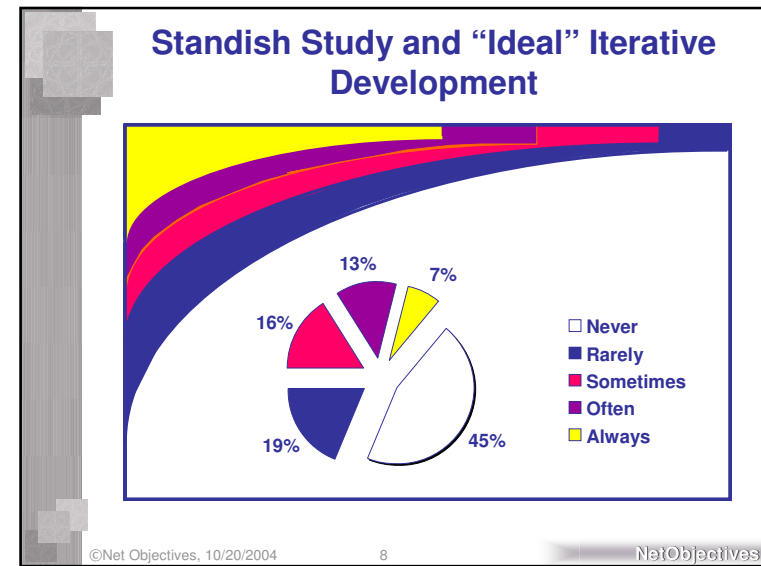
©Net Objectives, 10/20/2004 5 NetObjectives



## So, the "Ideal" project would...

- convert effort you would spend unwisely
  - Non-essential functionality
  - Doing the "wrong thing"
- into developing things you don't know about yet
  - New functionality
  - Additional "robustness"
  - As-yet undiscovered bugs
- We're not so good at looking into the future, so this requires some sort of iterative process...

©Net Objectives, 10/20/2004 7 NetObjectives



## Summary of Project Management

- **Goal:** Provide a suitable solution for users...That consists of quality code...And doesn't cost too much
- **Ideal:** Convert effort you would spend unwisely into developing things you don't know about yet
- **Common Strategy:** Get your money, then spend it wisely
- **Implementation:** We claim this *requires* an agile process

## The Essence of the Practice of Agility is Iteration, Validation and Feedback

- User / Customer Feedback Loop allows us to Satisfy our Users



## What's Important About Agility (for our purposes, right now...)

- Agility is about doing everything in small, validated chunks
  - Validate that chunks are the right ones
  - Validate that we have accomplished them
- So, a process is amenable to agility if it can be decomposed into small chunks, each of which can be validated
- This is how we do our use cases, as we'll see in a moment

## “good” Team Philosophies, leading to Agility

- **One Bite at a Time:** reminds us to do things a little at a time, with planning, validation, and management of the pieces.
- **Validation Centricity:** reminds us that the activities of validation, verification and test are “more important” than those of analysis, design, and construction; and that we must actively look for things that cause us to change.
- **Don't Do What you Don't Need:** work on those things with the most value; avoid analysis paralysis, and keep moving
- **Quality Can't be the Variable:** When balancing the three main variables (scope, time, quality), quality can't be the one that slips
- **Let the Product Lead:** decisions must be based on the product, not documented plans, analyses, requirements, or designs.

## Doing the Work in an Agile Project



## Managing the Building of Product

- **Work Backlog:** an evolving, prioritized queue of business and technical functionality that needs to be developed into a system (Source: Scrum)
- **Story:** Describes some item of value to a user or product owner (Source: XP, Cohn)
- **Task:** Well-defined unit of work in the process that provides management with visible checkpoints into the status of the project. Tasks have readiness criteria and completion criteria. [Jones - IBM] (Source: SEI:SE-CMM)

## Example Stories

- Description: Shop for Round Trip Flights
  - Size: Big
  - Validation: Functional test each Scenario
- Description: Round Trip, single passenger, no luggage, no seat selection, one leg
  - Size: Large
  - Validation: Functional Testing
- Description: Add the Cello
  - Size: Small
  - Validation: Functional Testing
- Description: Help Marketing Prepare materials for the trade show
  - Size: Large
  - Validation: Inspection by Marketing

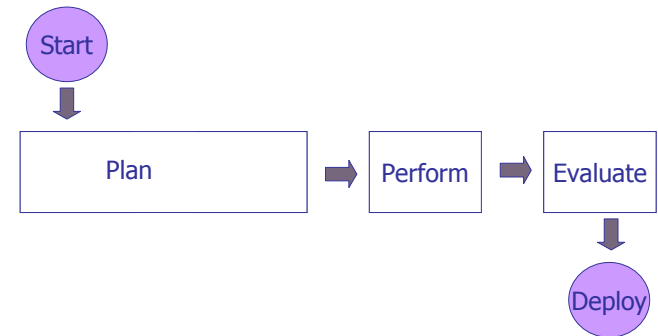
## Example Tasks

- Description: New build script
  - Estimate: 6 hr
  - Owner: Sue
  - Exit Criteria/Verification: Build succeeds, Joe will inspect results
- Description: Add the "multi-passenger" check-box to the EntryScreen
  - Estimate: 4 hr
  - Owner: Joe
  - Exit Criteria/Verification: Unit tests written and passed
- Description: SQL Training
  - Estimate: 2 days
  - Owner: Sue/Joe
  - Exit Criteria/Verification: training completed

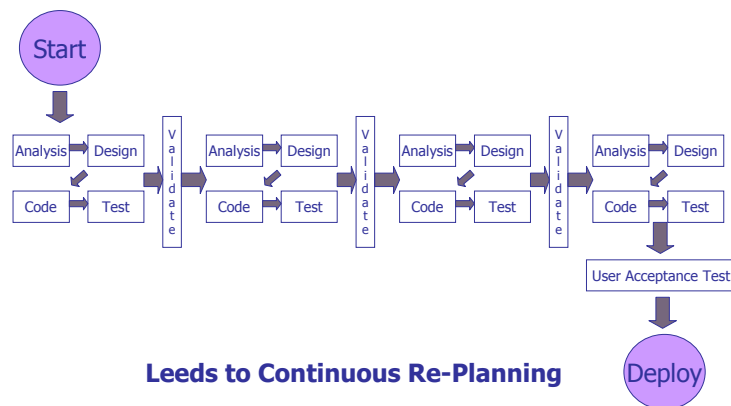
## Time Boxing

- Time Boxing means we have set start and end times for development cycles and a fixed agenda.
  - Scrum suggests 30 days.
  - XP suggests 1-4 weeks
  - RUP suggests 2-6 weeks.
- Prevents:
  - analysis paralysis
  - wild swings in effort over the course of the project
- Whatever the cycle, the customer selects the most important stuff to do and the developers get as much of that done as they can in that time.
- In other words: incremental, iterative development

## Instead of This



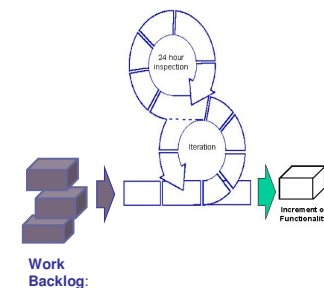
## We Need This



## The Basic Development Loop

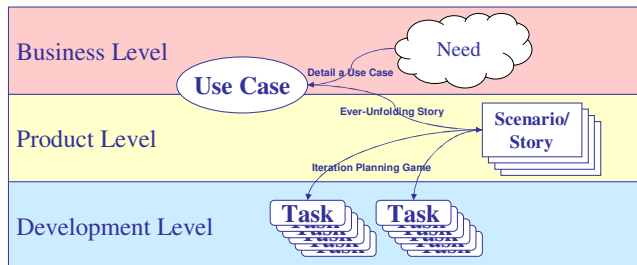
- For Each Iteration you:
  - **Plan**
    - What's the next stuff to do?
      - Analysis Work
      - Design Work
      - Implement some functionality
      - Write Functional Tests
      - Run Functional Tests
    - The team decides how much it can do
  - **Perform** – You do it
  - **Evaluate** – You evaluate both the Product and the Process
- Repeat...
- You manage this iteration with the Work Backlog

### Simplified Scrum (an agile process)



## Needs, Use Cases, Stories, Tasks – Creates Work We Need to Manage

- Here's what I recommend for the major steps in moving from a User's need to a collection of Developer Tasks



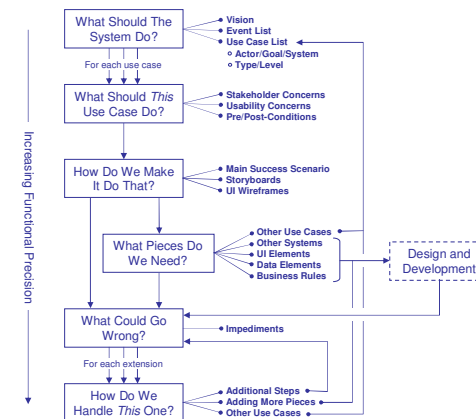
## Agenda

- Agility
  - Use Cases Driven Agile Development
    - Planning the Agile Project
- Use Case Development
- Summary

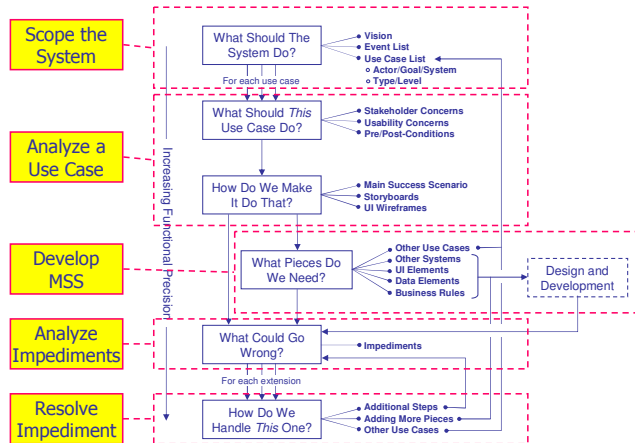
## Two Views of Use Cases

- Analytic View:** a conversation between an Actor and a System to Achieve a Goal
  - Used to elicit, document, and validate functional requirements of the system
- Development View:** A Bucket of Scenarios
  - Used as starting point for generating functional stories and tasks
  - Each scenario can be “unfolded” into the software elements that are actually developed and delivered

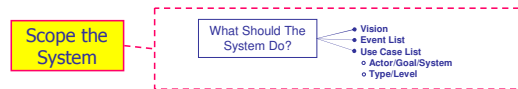
## Summary of Use Case Development



## Stories Based On Use Cases



## First, Get the Use Cases

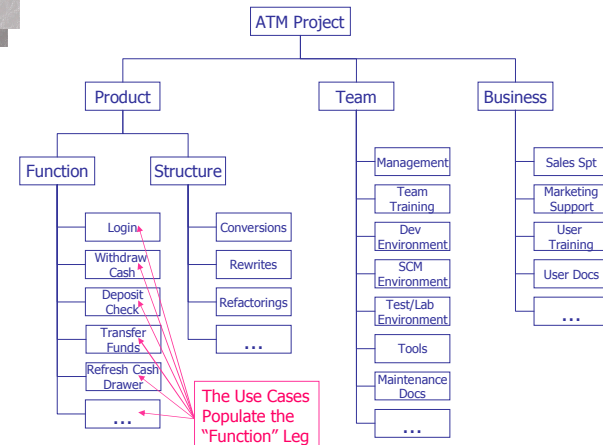


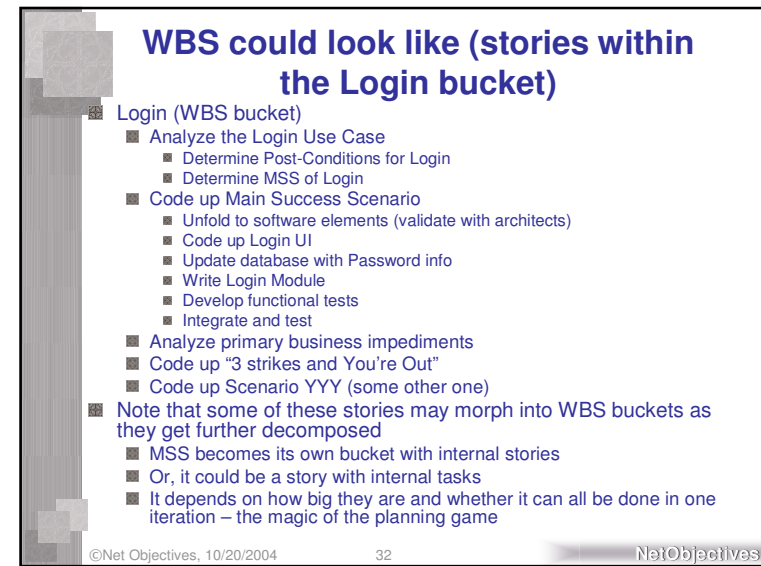
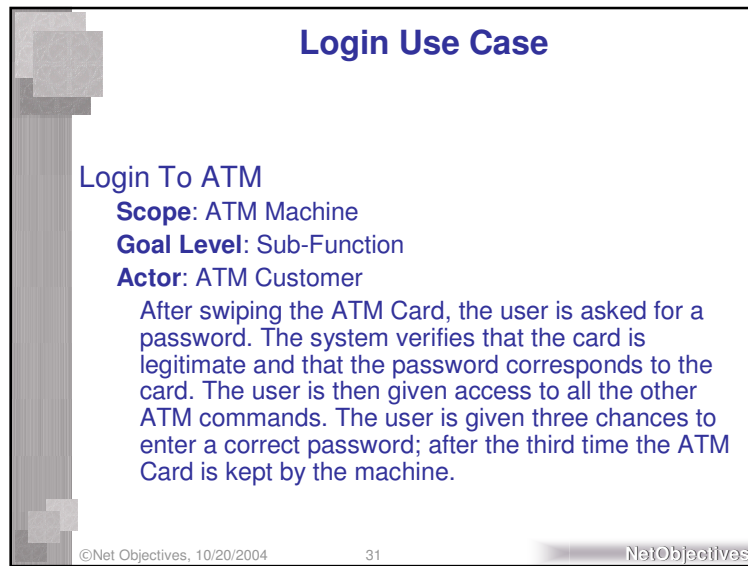
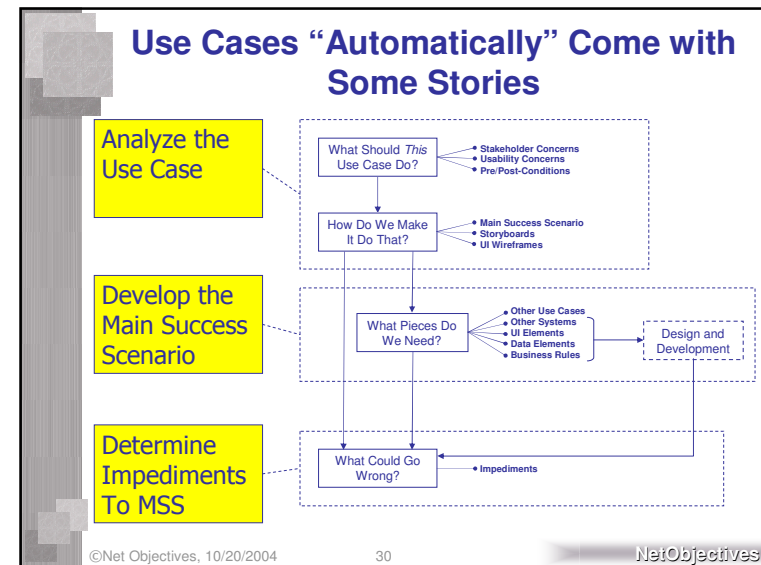
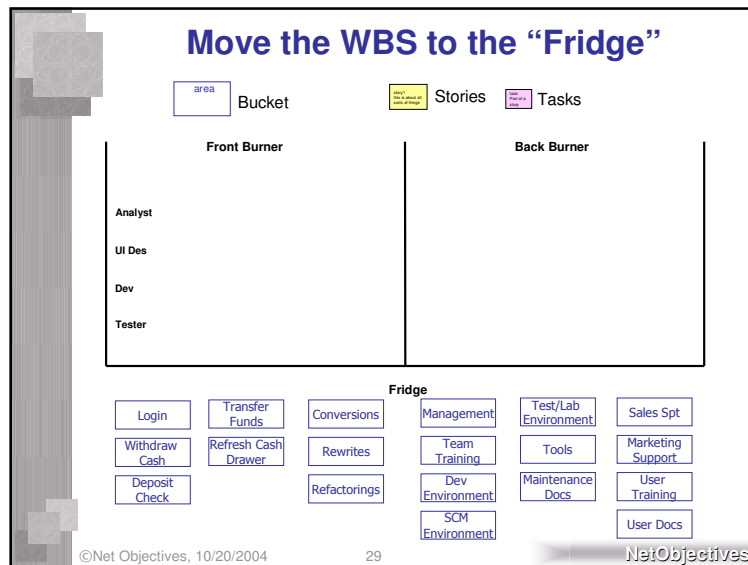
Actor	Event (optional)	Use Case Name
User	Use System	Log In
Customer	Wants Cash	Withdraw Cash
	Payday	Deposit Check
		Transfer Funds
Teller	Cash Drawer Empty	Refresh Cash Drawer
		Make Reports
Service Man		Maintain Machine

## The Work Breakdown Structure

- **Work Breakdown Structure:** A deliverable-oriented grouping of project elements which organizes and defines the total scope of the project. Each descending level represents an increasingly detailed definition of a project component. Project components may be products or services. (Source: PMI:PMBOK )
- **Breakdown Methods:**
  - Most standard projects use a work breakdown structure organized around types of activities, like analysis, design, code, test, project management, GUI Dev, DB Dev, etc.
  - This is because most project's management focuses on money or people, not the product – remember waterfall is a strong attractor
  - We do it differently

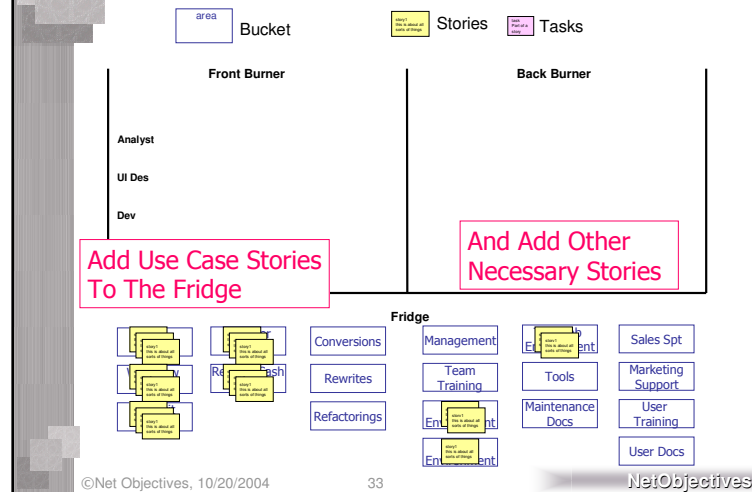
## Populate WBS Structure



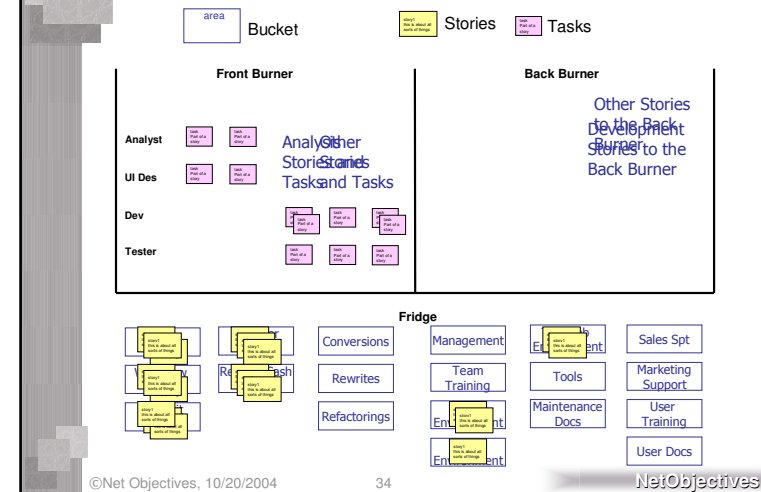


# NetObjectives

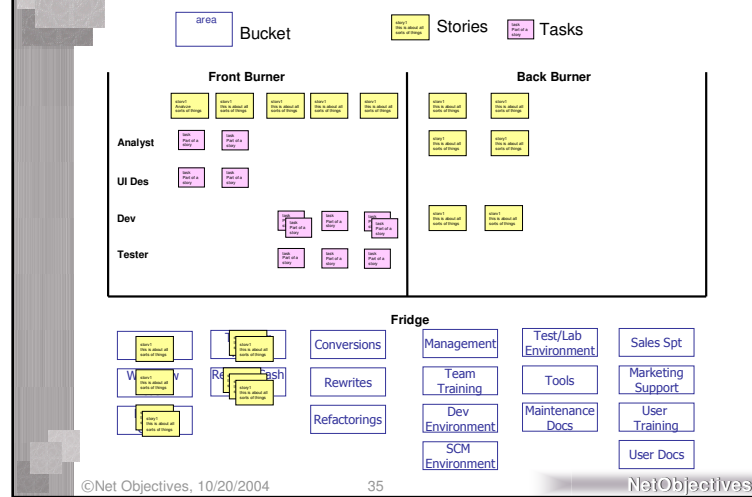
## Adding Stories to the "Fridge"



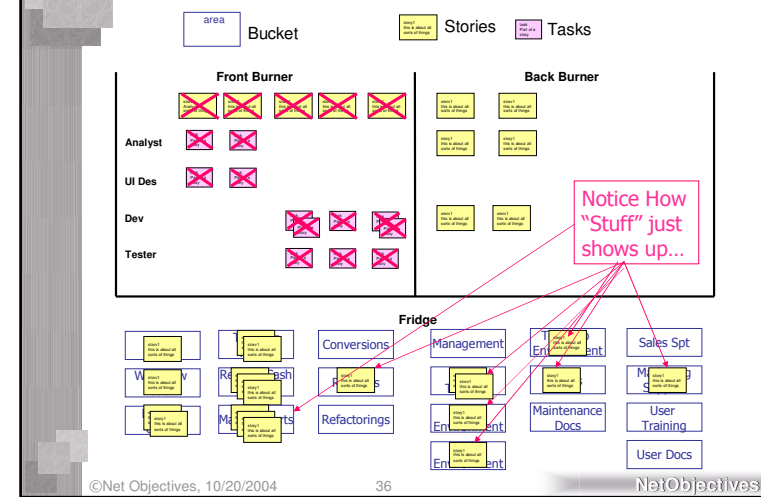
## Analyze and Prioritize The Stories



## Our Plan for the First Iteration



## Now Do the Work



# NetObjectives

## Note That...

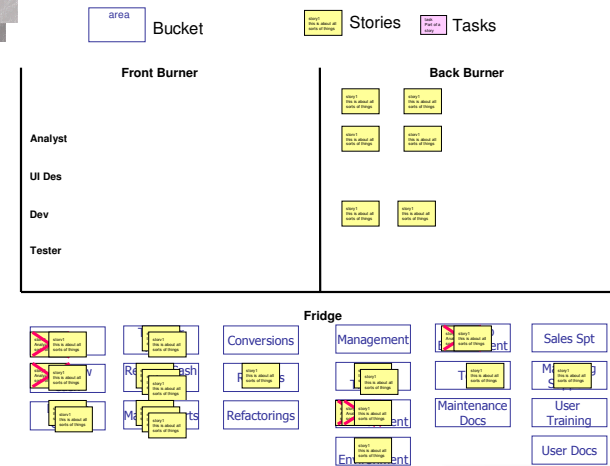
- The Use Case work we did was analysis work
  - What Should the Use Case Do?
  - How Do We Make it Do That?
- The Development of the MSS (for these two use cases) is on the Back Burner
- And the Analysis of what could go wrong is in the Fridge
- It gets interesting in the next Iteration...

©Net Objectives, 10/20/2004

37

NetObjectives

## We Clean up To Prepare...

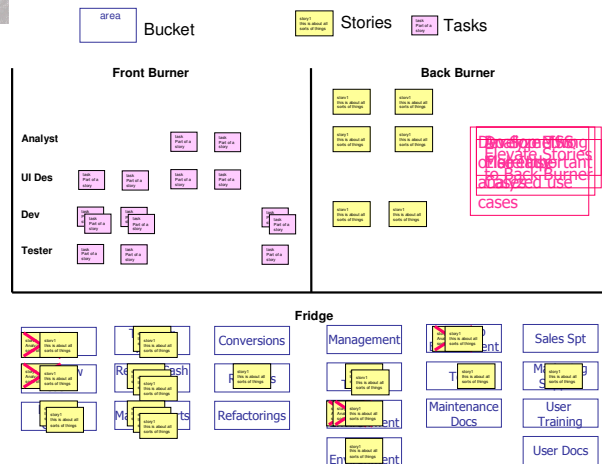


©Net Objectives, 10/20/2004

38

NetObjectives

## Plan for the Second Iteration

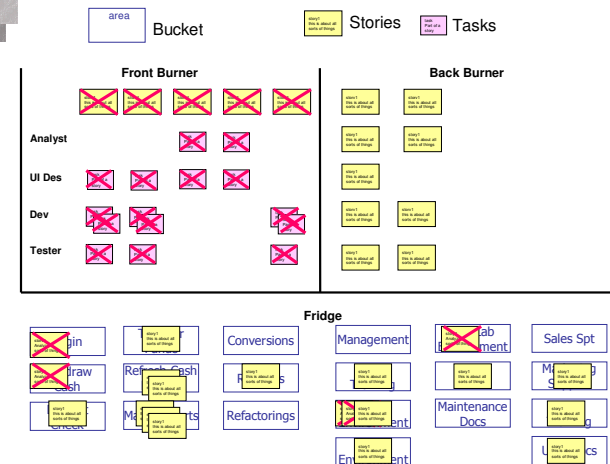


©Net Objectives, 10/20/2004

39

NetObjectives

## We Now Do the Second Iteration



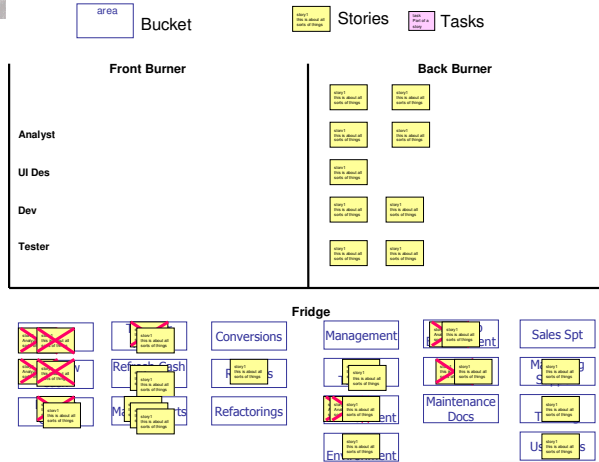
©Net Objectives, 10/20/2004

40

NetObjectives

# NetObjectives

## We Clean up To Prepare...



©Net Objectives, 10/20/2004

41

NetObjectives

## Note That...

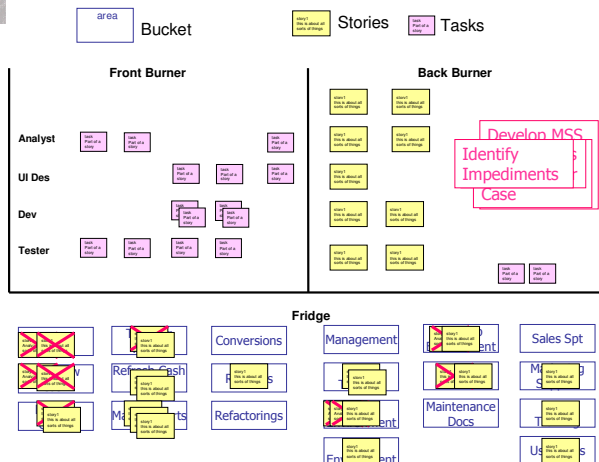
- For Our Use Cases
  - We have developed the MSS for two of them
  - We have analyzed two more of them
- Let's plan for iteration 3

©Net Objectives, 10/20/2004

42

NetObjectives

## Plan for the Third Iteration



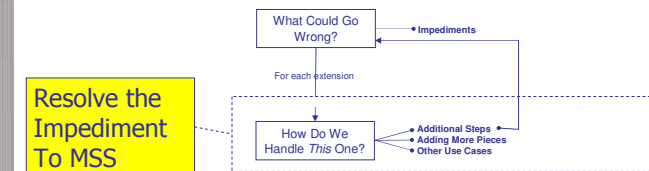
©Net Objectives, 10/20/2004

43

NetObjectives

## Finally, in this iteration

- We have some new stories showing up for our use cases, as impediments are identified
- We have a new kind of story



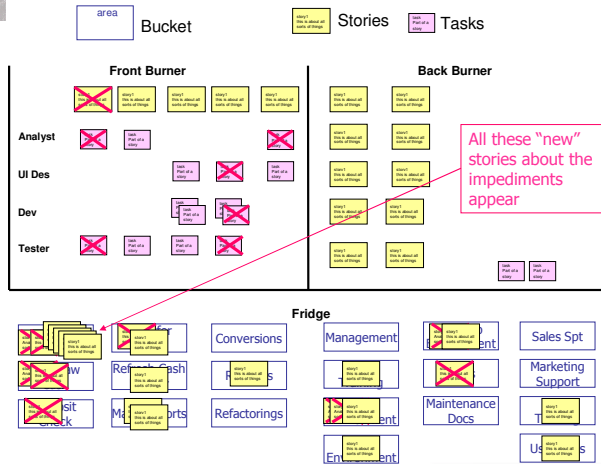
- And this is where the fractal nature of use case development shows up, big time...

©Net Objectives, 10/20/2004

44

NetObjectives

## Now, We Do the Third Iteration



©Net Objectives, 10/20/2004

45

NetObjectives

## And So On...

- As you can see, the in this example the use cases just keep on going
  - It took two iterations for the first use cases' Main Success Scenarios to be developed
  - Then we started finding impediments and have a bunch of new stories to work on...
- This is the nature of Agility
  - You keep finding new stuff
  - You keep prioritizing what you want to work on
  - You continually produce the best product you can for the effort you spend

©Net Objectives, 10/20/2004

46

NetObjectives

## Agenda

- Agility
- Use Cases Drive Agile Development
- Use Case Development
  - Overview
  - What Does the System Do?
  - What Does *This* Use Case Do?
  - How Do We Make It Do That?
  - What Pieces Do We Need?
  - What Could Go Wrong?
  - How Do We Handle That?
- Summary

©Net Objectives, 10/20/2004

47

NetObjectives

## Use Case Precision Levels

- Level 0: Name of the Use Case
- Level 1: Post conditions for the use case
- Level 2: MSS
- Level 3: Impediments / Errors / Exceptions
- Level 4: Handling these

©Net Objectives, 10/20/2004

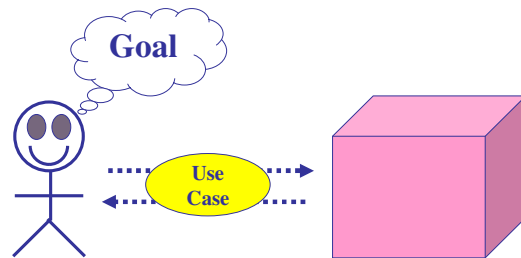
48

NetObjectives

# NetObjectives

## The Analytic View

- A Use Case describes a conversation between Actor(s) and System(s) in order to achieve Goal(s)



## Example: Mary Gets “Fast Cash”

Mary, taking her two daughters to the day care center on the way to work, drives up to the ATM, runs her card across the card reader, enters her PIN code, selects FAST CASH, and enters \$35 as the amount. The ATM issues a \$20 and three \$5 bills, plus a receipt showing her account balance after the \$35 is debited. The ATM resets its screens after each transaction with FAST CASH, so Mary can drive away and not worry that the next driver will have access to her account. Mary likes FAST CASH because it avoids the many questions that slow down the interaction. She comes to this particular ATM because it issues \$5 bills, which she uses to pay the day care provider, and she doesn't have to get out of her car to use it.

This use case is a very casual one, just a story, really. We often use more ceremony in our use cases, and like the ones found in Cockburn's "Writing Effective Use Cases" book.

## Why the Use Case is Important in Analysis

- We deliver solutions, not code
- So we need to focus on solutions, not code
- The use case describes how our product provides a solution
- The use case is a good way for everybody to communicate about what the system will do, and how it should be built

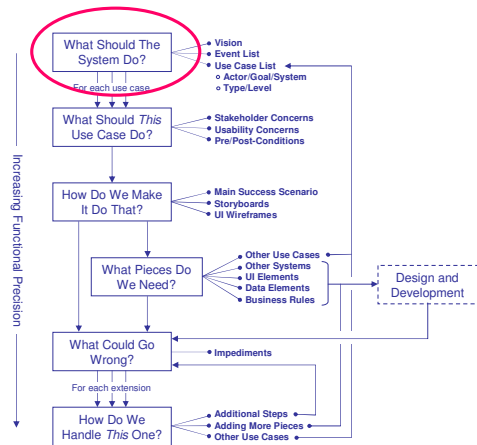


## So, Let's Develop Those Use Cases...



# NetObjectives

## What Should the System Do?



©Net Objectives, 10/20/2004

53

NetObjectives

## Samples

<i>For</i>	Small Consulting Group
<i>Who</i>	Is using paper for everything
<i>The SCG Application</i>	Is a web-based tool
<i>That</i>	Handles payroll, billing, and scheduling of Consultants
<i>Unlike</i>	Anything that we can find...
<i>Our Product</i>	Will enable us to practice what we preach

Event	Response
Course Completed	Invoice Client
Client asks for Course	Schedule course
End of Day	Employee fills our Timesheet
Payment Received from Client	Payment logged
Employee needs time off	Schedule Vacation Time

©Net Objectives, 10/20/2004

54

NetObjectives

## Use Case List

- An Actor is the entity (person) that actually interacts with the System
- The Use Case Name represents the Actor's goal
- For ATM Machine:

Actor	Event (optional)	Use Case Name	Level (optional)
User	Use System	Log In	Internal
Customer	Wants Cash	Withdraw Cash	
	Payday	Deposit Check	
		Transfer Funds	
		Go To Dinner and a Movie	Contextual
Teller	Cash Drawer Empty	Refresh Cash Drawer	
		Make Reports	
Service Man	Out of Receipt Paper	Replenish Receipt Paper	Internal
		Maintain Machine	
Bank Robber		Steal Money	
		Obtain Password	Internal
		Move to Argentina	Contextual

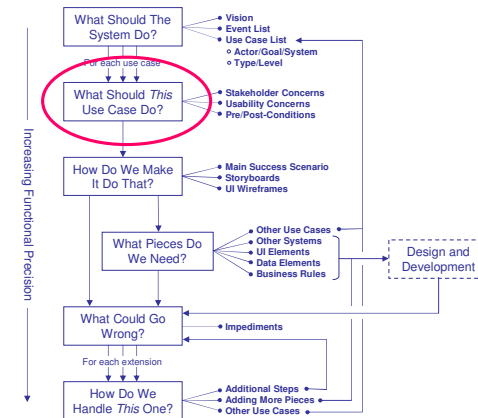
Out of Scope for This Talk

©Net Objectives, 10/20/2004

55

NetObjectives

## What Should This Use Case Do?



©Net Objectives, 10/20/2004

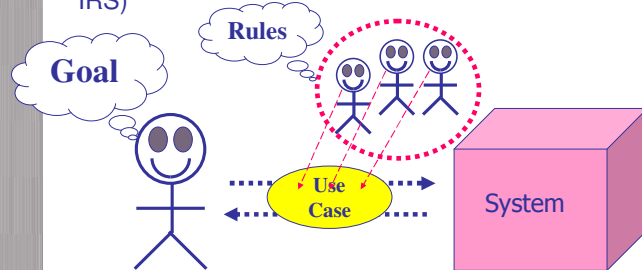
56

NetObjectives

# NetObjectives

## Stakeholder

- Somebody/thing that cares how the use case is executed – provides constraints and rules
- Not all stakeholders interact with the system (eg. The IRS)



©Net Objectives, 10/20/2004

57

NetObjectives

## Stakeholders and Interests

- A Stakeholder is someone or something that has an interest in the behavior of a use case
  - A Stakeholder's interests are usually captured in Business rules or Additional Goals
  - These interests usually concern the execution of the use case, not the "totality" of the use case
  - Often found / discovered late in the process...
- For ATM Machine

Stakeholder	Interest
Bank Owner	Assure Customer Properly Identified
	Don't inconvenience Customers too much
	Collect any (and all) applicable fees
FDIC	Follow All Banking Regulations
	Completely Track all Transactions

©Net Objectives, 10/20/2004

58

NetObjectives

## Usability Issues Also Show Up

- The way I see it (as a developer, not a usability expert), there are three usability issues:
  - Flow between pages/screens/dialogs/etc, which is a use case issue
  - Interaction on a page/screen/dialog/etc, which may or may not be a use case issue
  - Look and Feel, which is definitely not a use case issue
- Unfortunately, these three issues are tightly coupled, which means that usability and use case design are tightly coupled
- Therefore, we often need usability analysis here

©Net Objectives, 10/20/2004

59

NetObjectives

## Example: Withdraw Cash

### Use Case: Withdraw Cash from ATM

**Context of Use:** ATM Customer wishes to withdraw money from ATM while executing the *Use ATM Machine* Use Case.

**System:** ATM Machine

**Level:** System (Sea Level, User Goal)

**Primary Actor:** ATM Customer

### Stakeholders and Interests:

*ATM Customer* -- wants cash returned and ATM Card returned. Wants a receipt. Wants to have appropriate amount of money deducted from account. Wants to pay as little fee as possible.

*Bank Manager* -- wants to have appropriate amount of money deducted from account. Wants as much fee as possible.

*Bank Regulator* -- wants to make sure all regulations are complied with

**Precondition:** ATM Customer is logged on

### Postconditions:

**Success:** Money and receipt dispensed to ATM Customer. ATM Card returned. Customer Account debited appropriate amount of money. Appropriate fees collected. All banking regulations adhered to.

**Minimal:** Transaction logged to completion or point of failure

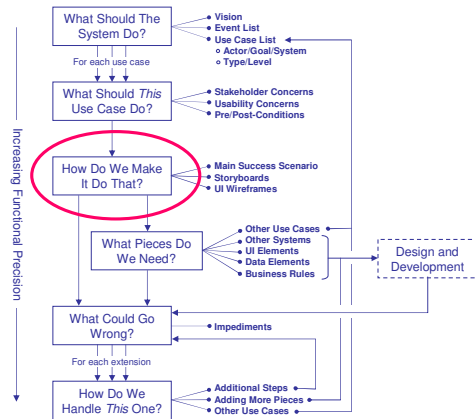
**Trigger:** ATM Customer selects "Withdraw Cash" option at main menu

©Net Objectives, 10/20/2004

60

NetObjectives

## How Do We Make It Do That?



©Net Objectives, 10/20/2004

61

NetObjectives

## The Main Success Scenario - Definition

- A top-to-bottom description of one easy-to-understand and fairly typical scenario in which the goals are delivered and the stakeholder's interests are satisfied
- The "Sunny Day" scenario – showing that the problem can be solved, and giving one way to do it
- Often seen as the "guts" of the use case – the initial design of how the system will satisfy its responsibilities

©Net Objectives, 10/20/2004

62

NetObjectives

## MSS as a Series of Steps

- Each Step is one of:

- An Interaction
- A Validation
- An Internal Change

- Example: Withdraw Cash

1. The ATM Customer selects an account to withdraw from, and enters an amount to withdraw, in multiples of \$20.
2. The ATM Machine verifies that there is enough money to dispense.
3. The ATM Machine notifies the Main Banking System of the Customer Account and amount being withdrawn
4. The Main Banking System returns the new balance and (if appropriate) the fee amount.
5. The ATM Machine dispenses the cash and updates the receipt.
- \* The ATM Machine logs each step of this transaction.

©Net Objectives, 10/20/2004

63

NetObjectives

## Some Guidelines for the MSS

- Keep It Short (< 10 steps)
- Keep It Simple
  - Subject => verb => direct object => prepositional phrase
  - "The system deducts the amount from the account balance."
- Keep It Clear
  - Know who has the ball, and what he's doing with it
- Keep It Positive
  - Assume success as you go (no "if" statements, validate instead)
- Use These Tricks:
  - Looping ("Do steps X-Y until Z")
  - Other Systems ("Get the ... from the Inventory System")
  - Timing ("... within 5 seconds")

©Net Objectives, 10/20/2004

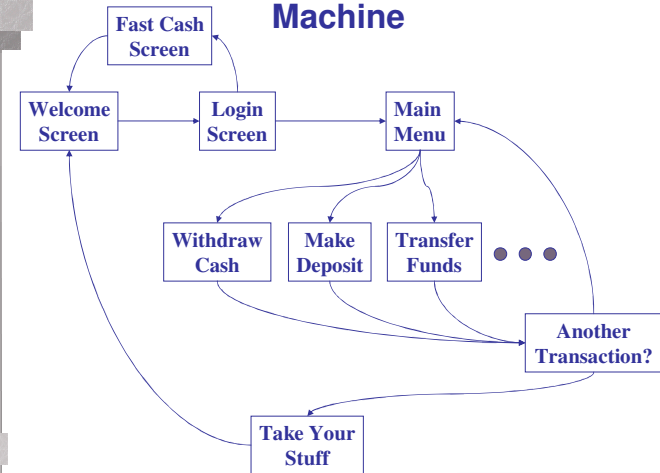
64

NetObjectives

## We Validate Four Ways

- Is it useful?
  - Did it satisfy its post-conditions?
- Is it usable?
  - May require storyboards and wireframes to validate
- Is it buildable?
  - Must verify with developers that fits within architectural scheme, etc
- Is it testable?
  - Can we envision the test scenarios we need to test it?

## Unfolded Story Board for ATM Machine



## UI Wireframes For Login

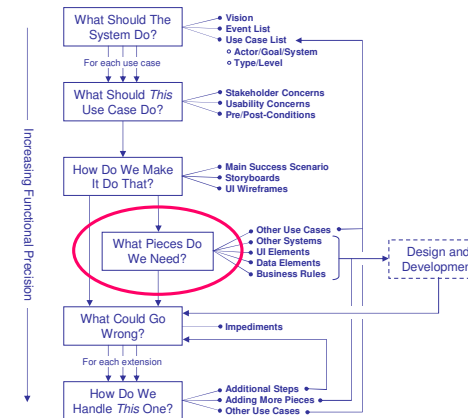


Old-Style Kiosk



Touch-Screen Kiosk

## What Pieces Do We Need?

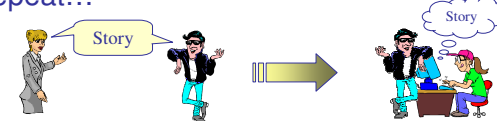


## We “Unfold” a Scenario

- Once we have a validated scenario we:
  - Figure out what data is needed
  - Figure out what UI is needed
  - Figure out what other Systems / Subsystems are needed
  - Figure out what Business Rules are in play
- Give everything Intention Revealing Names
- Update the Use Case, providing traceability
- A Wiki is a very cool tool to use here

## Basic Strategy

- Validate what we “know”
- Encapsulate what we don’t know by labeling it with an “Intention Revealing Name”
- Investigate, model, etc, these “new things”
- Repeat...



- Main Difficulties (analysis paralysis)
  - Digging too deep too fast
  - Not having right people to validate

## Use Case: Withdraw Cash from ATM

### Main Success Scenario:

1. The ATMMachine displays the WithdrawCashScreen
2. The ATM Customer selects the WithdrawalAccount, from, enters the WithdrawalAmount, in multiples of \$20.
3. The ATMMachine verifies that there is enough money to dispense in its CashDispenser
4. The ATMMachine notifies the MainBankingSystem of the CustomerAccount and WithdrawalAmount, and the MainBankingSystem returns the new AccountBalance and the AssessedFee amount (if any)
5. The ATMMachine dispenses the WithdrawalAmount and updates the TransactionReceipt

\*The ATM Machine logs each step of this transaction.

## Unfolded Data Element

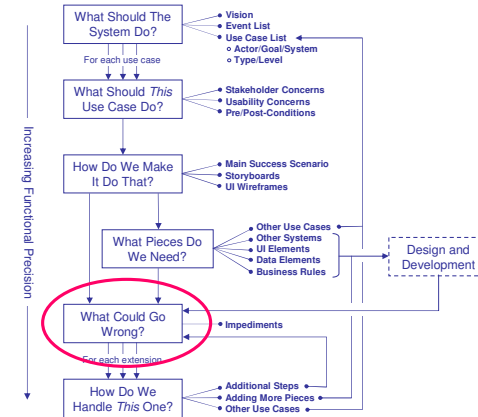
### CustomerInfo

- Name
- ID
- Address
- Phone
- Fax
- Email
- Typically moves from logical view (here), to physical design as we continue to move past analysis and into design. This just provides traceability by use of the IntentionRevealingName that is common throughout

## You Find Business Rules

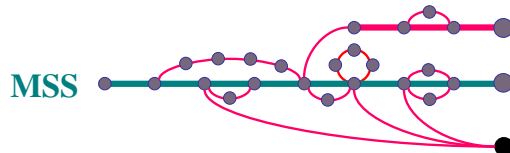
- Business Rules have the following attributes (Sue Burk):
  - Intention Revealing Name
  - Description - in business language
  - Rule Source ( a person, organization, or even an old computer program)
  - Rule Owner (role or person or organization to contact for changes in the rule)
- We may also need to know if the business rule is Static or Dynamic – how often does it change?

## What Could Go Wrong?



## Impediments

- Use Cases contain all the alternatives to attain a given goal, including failure alternatives
- In this formulation of use cases, those failures that prevent the MSS from being accomplished are called "impediments"



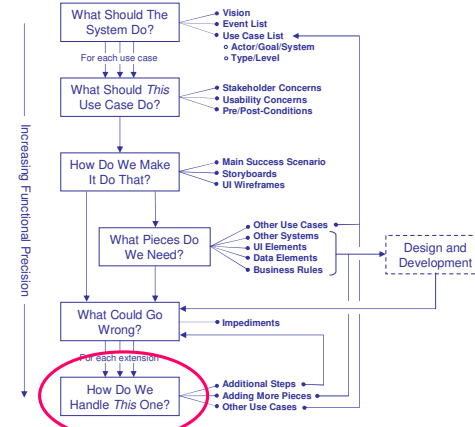
## Identify Impediments

- For Each Step in the MSS:
  - Brainstorm Conceivable Failures – don't expect to get them all
  - Could be of various kinds:
    - At business level
    - After Unfolding
    - After actually building (observed bug)
- Be Sure to Consider:
  - The Actor behaving incorrectly
  - Validation failing
  - Internal Failures in System
    - Expected failure modes
    - Abnormal failure modes
  - Critical Performance Failures

## Documenting Impediments

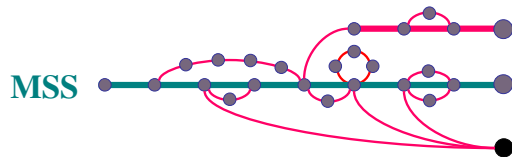
- You never want to obscure the MSS in your Use Case
  - Need it sitting there as a guiding light
- Document your Impediments separately
  - Numbering your MSS steps makes it easier
- Examples:
  - \*a. Network Goes Down
  - \*b. User walks away without Notice (time-out)
  - 1a. User enters non-multiple of \$20
  - 2a. Insufficient funds in ATM
  - 4a. Bank Computer does not reply
  - 5a. Out of Receipt Paper

## How Do We Handle *This* One?



## The Meat of the System

- The hardest part about any system is exception handling
- This use case formulation brings this problem right out in the open
- The users (or their surrogates) get to chime in on what should be done in response to impediments
- If you don't have it in the use case, then you're letting the developer decide...



## So, what happens?

- For each Impediment, you can:
  - Clean Up Mess and Bail Out (CUMABO)
  - Interact with the user for a few extra steps and reenter the MSS
  - Fork to a different Use Case
- Deciding which Impediments to handle, and how, is one of the biggest problems you will have
  - You don't know them all up front, so they always hit you unexpectedly
  - They take resources to solve...

## For Example...

- Say you have Impediment
  - 2a. Insufficient Funds in ATM
- Then do you:
  - Terminate the user and Shut Down the ATM?
  - Terminate the user and wait to see if you can satisfy the next guy?
  - Give as much money as you can?
  - Give the user the option to choose a smaller amount?
- These options have different costs and benefits, and what to do needs to be decided by the right people...

## Agenda

- Agility
- Use Case Development
- Integrating Use Cases into Agile Development
- Summary

## Two Views of Use Cases

- Analytic View: a conversation between an Actor and a System to Achieve a Goal
  - Used to elicit, document, and validate functional requirements of the system
- Development View: A Bucket of Scenarios
  - Used as starting point for generating functional stories and tasks
  - Each scenario can be “unfolded” into the software elements that are actually developed and delivered

## Two Views of Work

- Work Backlog
  - Organizing for development
  - Based on order of development / priority
  - Used to calculate Velocity
  - Most useful inside the team to track progress
- Work Breakdown Structure
  - Organizing for Content of Project
  - Based on decomposing the project elements
  - Used to find new stories and make sure have the ones we want
  - Can be used to calculate Earned Business Value

## Putting Them Together

- The Whole Team Concept
  - Customers and Developers working and planning together
  - Prioritizing analysis, design, development, test stories in one planning game
- Defining the work with Use Cases
- Using the Work Backlog inside the Team
  - Manage work being done
- Using the Work Breakdown Structure
  - Outside the team to advertise progress (EBV)
  - Inside the team to help prioritize and determine coverage

**Any Questions?**

**Please Fill Out Your Evaluations**