AM04

Conversations in Context – Using Use Cases on Agile Projects

Ian Spence

Chief Scientist, Ivar Jacobson International

ispence@ivarjacobson.com

IBM Rational Software Development Conference 2008

WHERE TEAMS ARE R-HEROES

Agenda

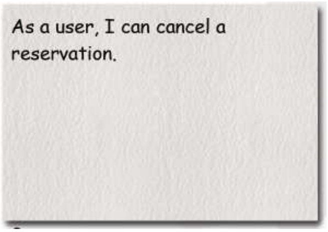
- What are agile teams looking for?
  - Cards, conversations, and confirmations
  - Knowing what to do and when it’s done
- Being agile with use cases
- Case Studies
- Demo
- Wrap Up / Getting Started

What are Agile Team's looking for?

- Something quick and lightweight
  - Placeholders for conversations (or even more formal requirements)
- Work items for the backlog
  - Small enough to tackle in an iteration
  - Provide value to the customer
- Definitions of done
  - Acceptance and other tests

The most common technique to use is user story cards.

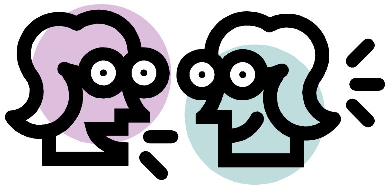
Cards, Conversations, and Confirmation



As a user, I can cancel a reservation.

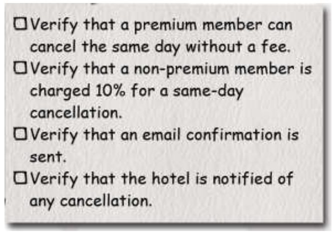
Card

- **Stories are traditionally written on note cards**
- **Cards may be annotated with estimates, notes, etc**
- **These are the placeholders for the conversations**



Conversation

- **Details behind the story come out during conversations with the product owner**



☐Verify that a premium member can cancel the same day without a fee.
☐Verify that a non-premium member is charged 10% for a same-day cancellation.
☐Verify that an email confirmation is sent.
☐Verify that the hotel is notified of any cancellation.

Confirmation

- **Acceptance tests confirm the story was coded correctly**

Adapted from: Ron Jeffries, “Essential XP: Card, Conversation, and Confirmation,” *XP Magazine*, August 30, 2001.

Agile teams want small pieces of work

- Stories support the customers and developers
- For customers easy to write and understand
- For developers small enough to be completed in an iteration

If a story is to big to implement then it is ripped up and replaced by a number of smaller stories.

Stories come in all shapes and sizes

User Story

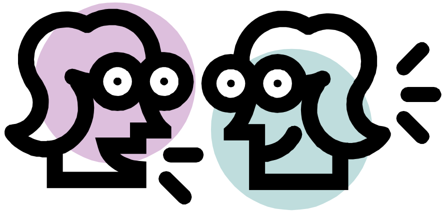
A description of desired functionality told from the perspective of the user or customer

Epic

A large user story

Theme

A collection of related user stories



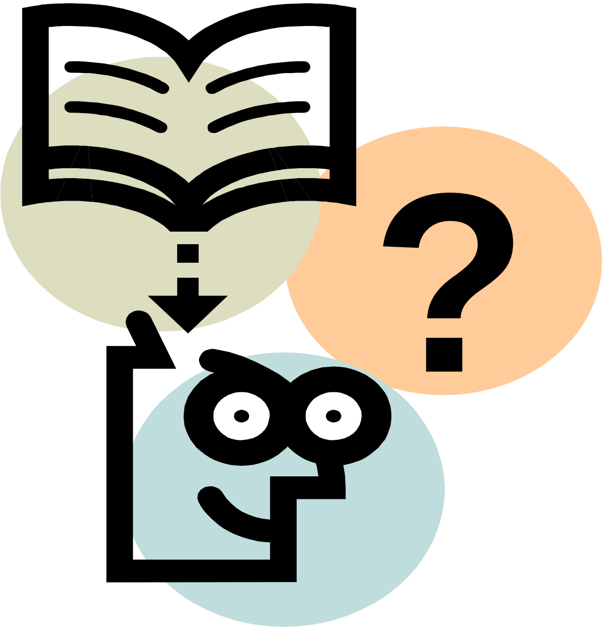
We often need a frame of reference before we can have a conversation.

What’s the context for the conversation?

Themes, Epics and User Stories – Adapted from An Introduction to User Stories by Mike Cohn

But where are the requirements?

- User stories aren't requirements
- Some people claim that the test cases are the requirements
- Some people maintain that the conversations are the requirements
- Some people add more formal requirements specifications to complement their stories



All requirements start as placeholders for conversations.

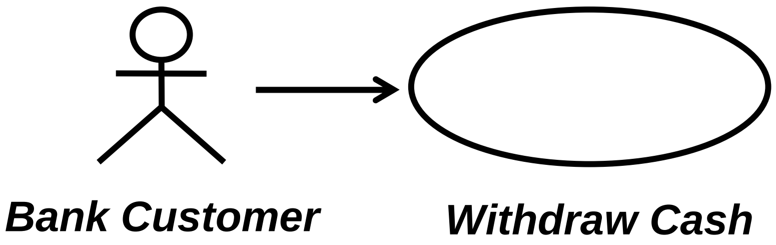
Agenda

- What are agile teams looking for?
- Being agile with use cases
  - A brief introduction to use cases and use-case modules
  - Selecting and prioritizing use cases
  - Using cards and backlogs
- Case Studies
- Demo
- Wrap Up / Getting Started

What is a Use Case?

A use case is the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system.

- Use cases are shown in UML diagrams

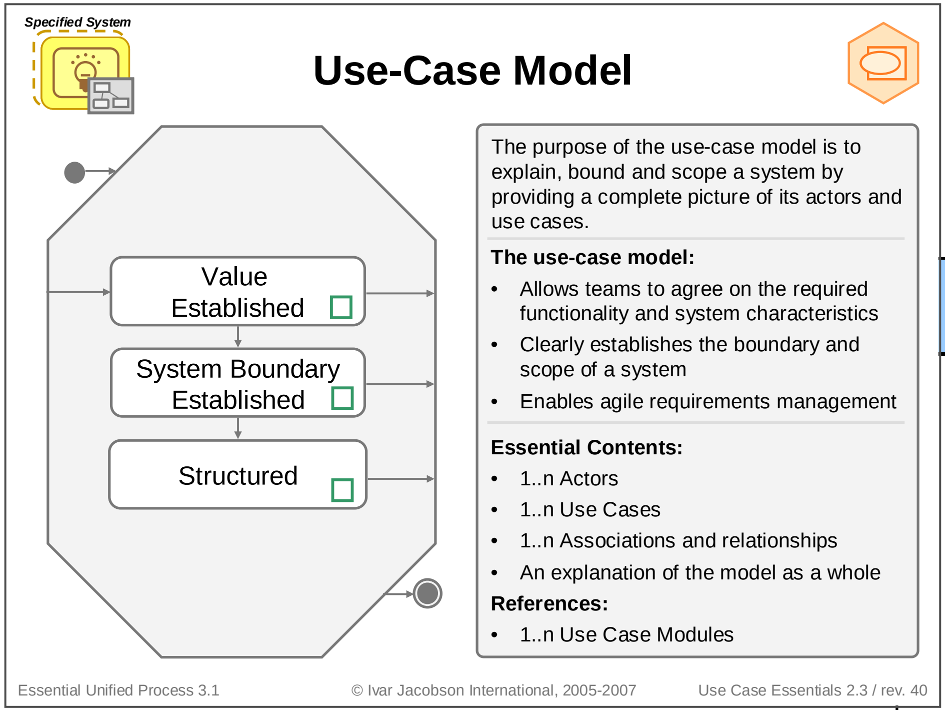


- Use cases are described in text
  - They tell the story of the interactions between actors and the system

Use-Case Modeling – A very simple idea.

To get to the heart of what a system must do, you should focus on who, (or what) will use it, and then look at what the system must do for them to help them achieve something useful.

What is a Use-Case Model?



Specified System

Use-Case Model

Value Established ☐
System Boundary Established ☐
Structured ☐

The purpose of the use-case model is to explain, bound and scope a system by providing a complete picture of its actors and use cases.

The use-case model:

- Allows teams to agree on the required functionality and system characteristics
- Clearly establishes the boundary and scope of a system
- Enables agile requirements management

Essential Contents:

- 1..n Actors
- 1..n Use Cases
- 1..n Associations and relationships
- An explanation of the model as a whole

References:

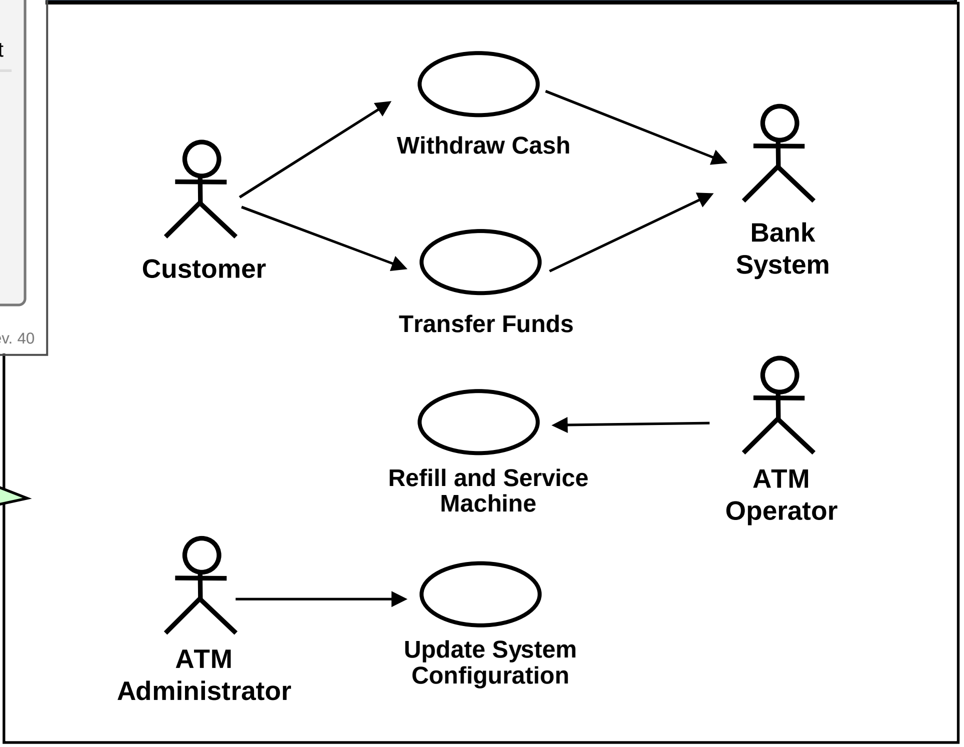
- 1..n Use Case Modules

Essential Unified Process 3.1 © Ivar Jacobson International, 2005-2007 Use Case Essentials 2.3 / rev. 40

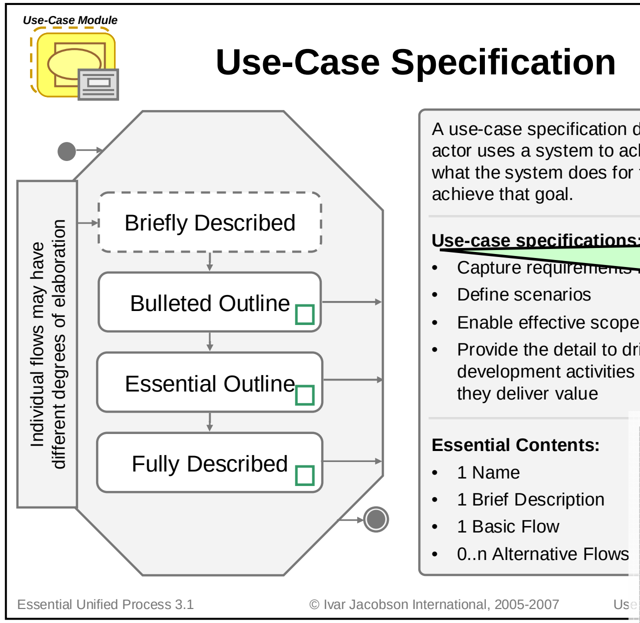
The use cases are our epics and themes.

The model provides a context for our conversations.

A Simple ATM System



Describing a Use-Case



Use-Case Module

Use-Case Specification

Individual flows may have different degrees of elaboration
Briefly Described
Bulleted Outline
Essential Outline
Fully Described

A use-case specification describes how an actor uses a system to achieve a goal and what the system does for the actor to achieve that goal.

Use-case specifications:

- Capture requirements in
- Define scenarios
- Enable effective scope management
- Provide the detail to drive the other development activities and ensure that they deliver value

Essential Contents:

- 1 Name
- 1 Brief Description
- 1 Basic Flow
- 0..n Alternative Flows

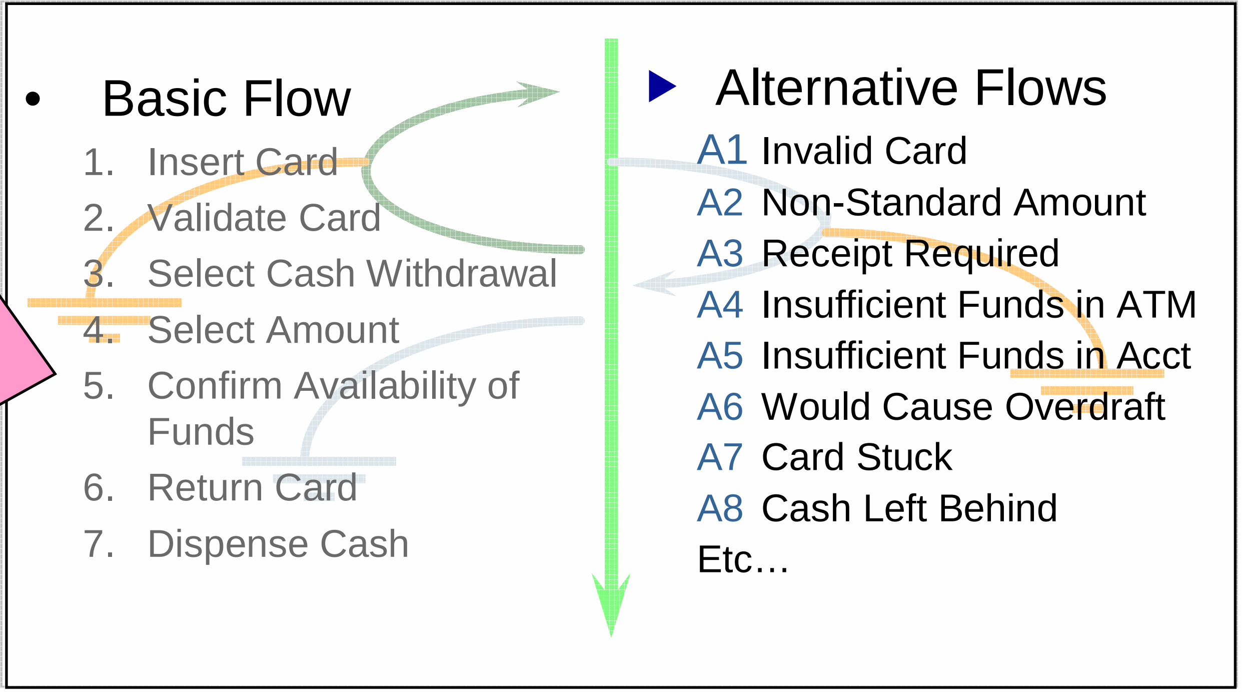
Essential Unified Process 3.1 © Ivar Jacobson International, 2005-2007 Use

Use cases can be described at different levels of detail.

They start very lightweight.

Use cases contain many different stories.

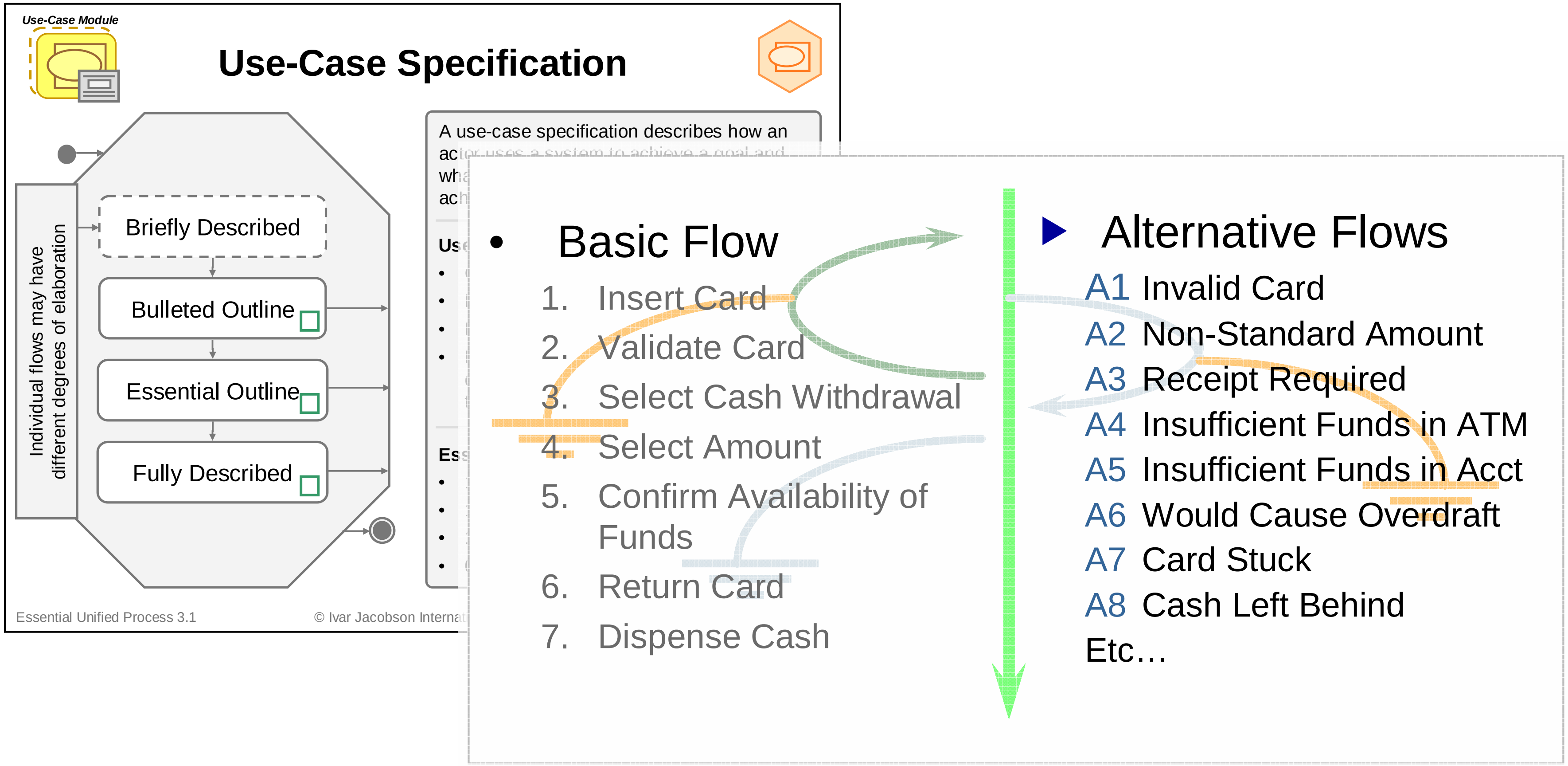
By outlining the use case we establish the context for a set of related conversations.



- Basic Flow
  1. Insert Card
  2. Validate Card
  3. Select Cash Withdrawal
  4. Select Amount
  5. Confirm Availability of Funds
  6. Return Card
  7. Dispense Cash

- Alternative Flows
  - A1 Invalid Card
  - A2 Non-Standard Amount
  - A3 Receipt Required
  - A4 Insufficient Funds in ATM
  - A5 Insufficient Funds in Acct
  - A6 Would Cause Overdraft
  - A7 Card Stuck
  - A8 Cash Left Behind
  - Etc…

Use Cases contain many requirements….



Use-Case Specification

A use-case specification describes how an

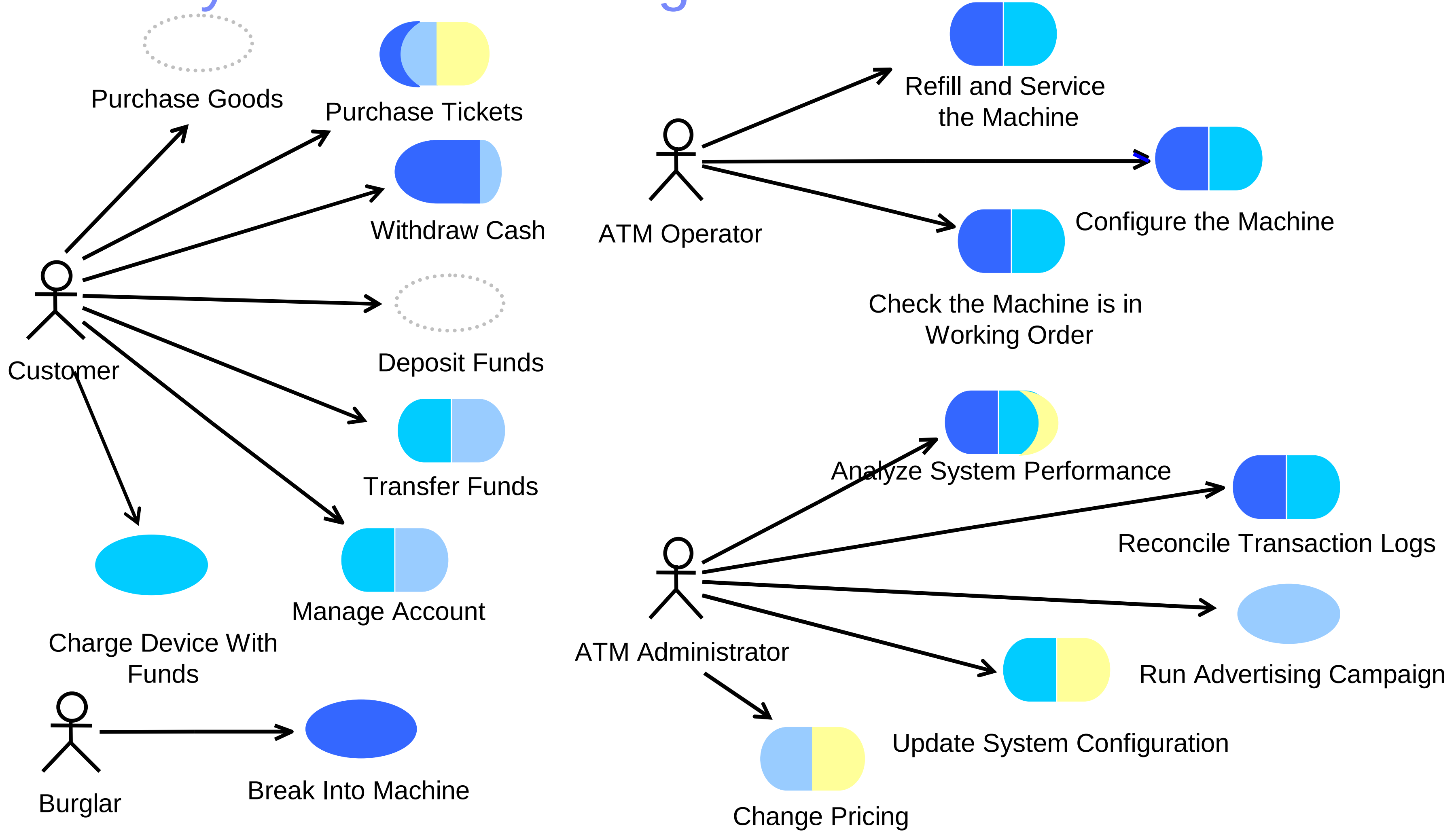
- Basic Flow
  1. Insert Card
  2. Validate Card
  3. Select Cash Withdrawal
  4. Select Amount
  5. Confirm Availability of Funds
  6. Return Card
  7. Dispense Cash

▶ Alternative Flows

A1 Invalid Card
A2 Non-Standard Amount
A3 Receipt Required
A4 Insufficient Funds in ATM
A5 Insufficient Funds in Acct
A6 Would Cause Overdraft
A7 Card Stuck
A8 Cash Left Behind
Etc…

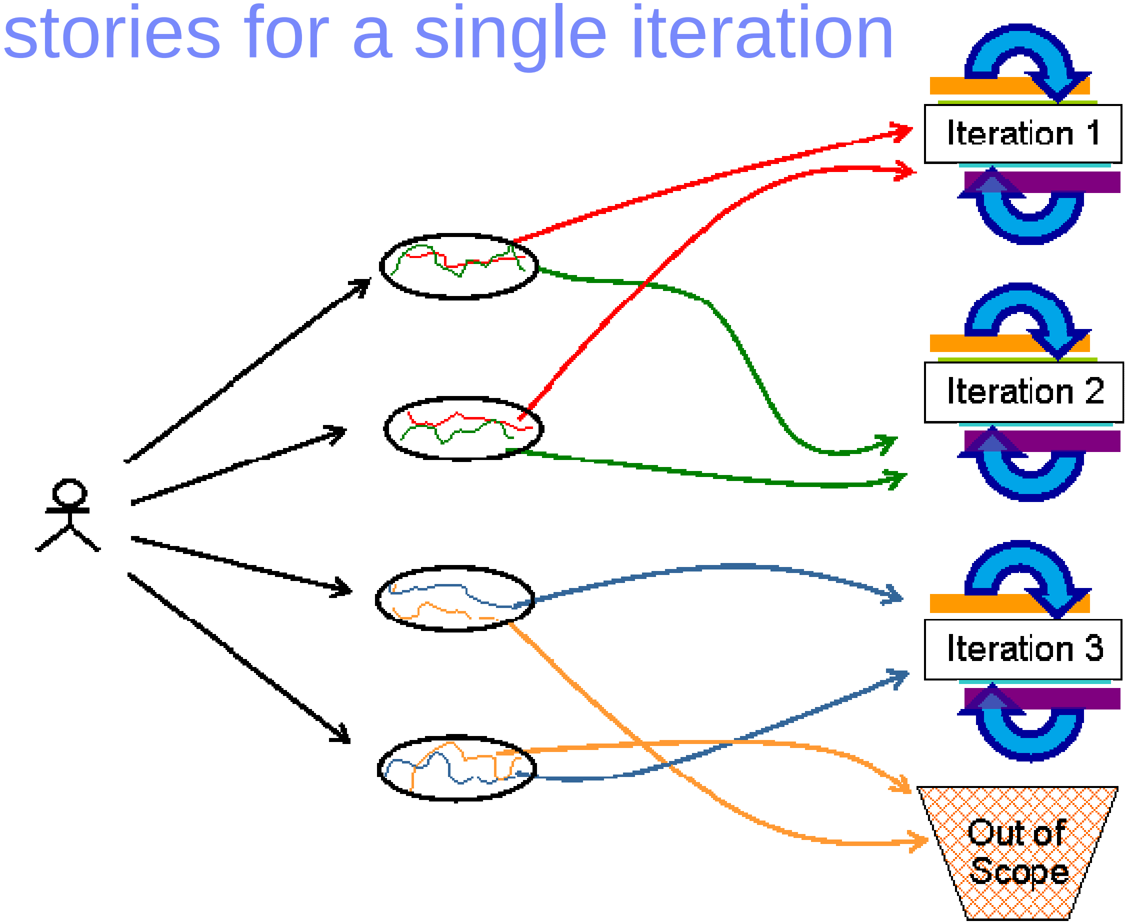
… often too many to code and test in one go.

Too many stories for a single release



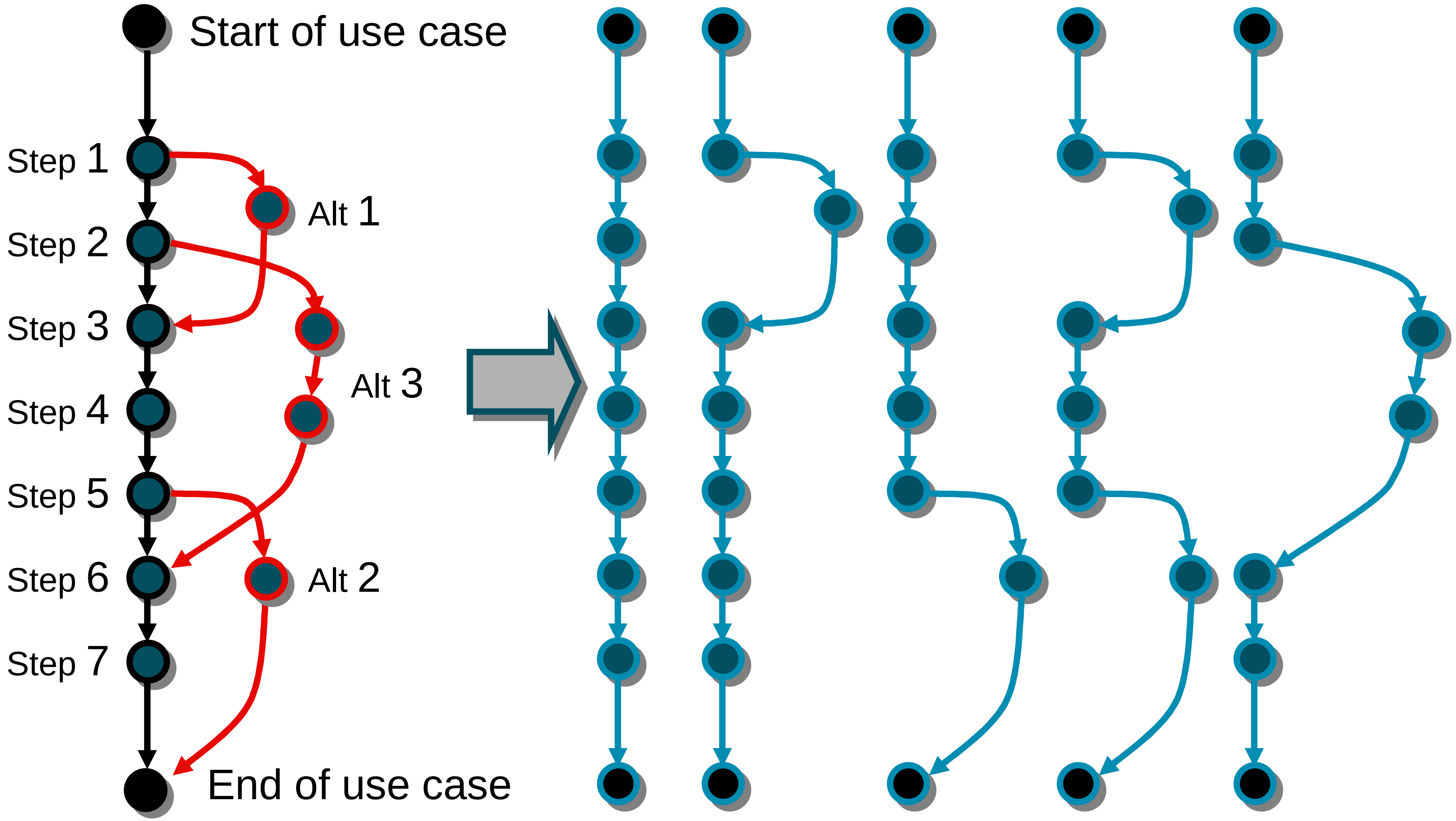
Release 1: ● , Release 2: ● ,
Release 3: ● , Release 4: ● , Out of Scope: ◌

Too many stories for a single iteration



Use cases provide the end-to-end threads required to set the objectives for the iterations

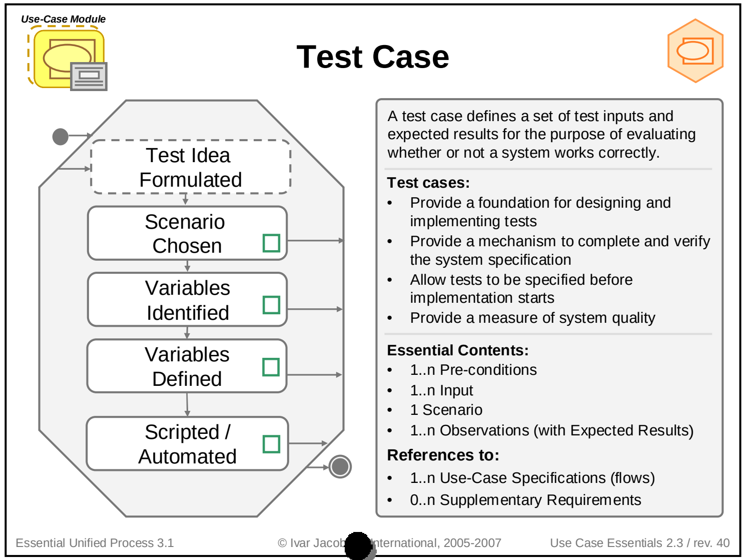
Use Cases Define Many Scenarios



1 Use Case

Many scenarios…

Use-Case based Test Cases


Use-Case Module
Test Idea Formulated
Scenario Chosen
Variables Identified
Variables Defined
Scripted / Automated

Test Case

A test case defines a set of test inputs and expected results for the purpose of evaluating whether or not a system works correctly.

Test cases:

- Provide a foundation for designing and implementing tests
- Provide a mechanism to complete and verify the system specification
- Allow tests to be specified before implementation starts
- Provide a measure of system quality

Essential Contents:

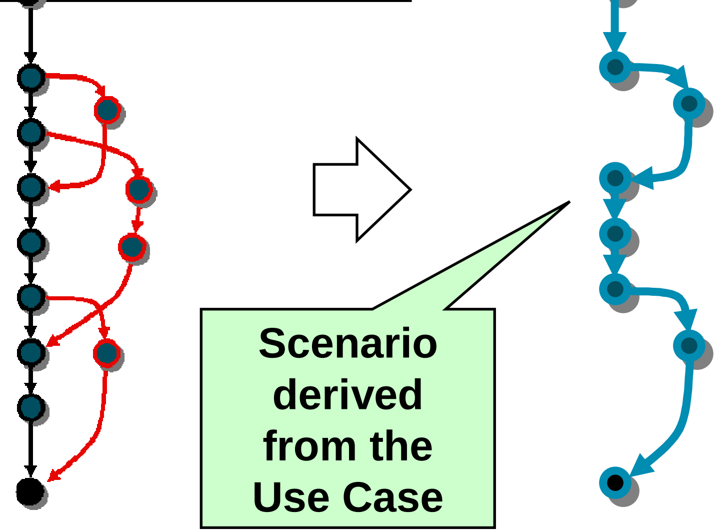
- 1..n Pre-conditions
- 1..n Input
- 1 Scenario
- 1..n Observations (with Expected Results)

References to:

- 1..n Use-Case Specifications (flows)
- 0..n Supplementary Requirements

Essential Unified Process 3.1 © Ivar Jacobson International, 2005-2007 Use Case Essentials 2.3 / rev. 40

- Set of **inputs** and **expected results** for the purpose of evaluating whether or not a system correctly implements a specific **scenario**
- Allow tests to be specified before implementation starts

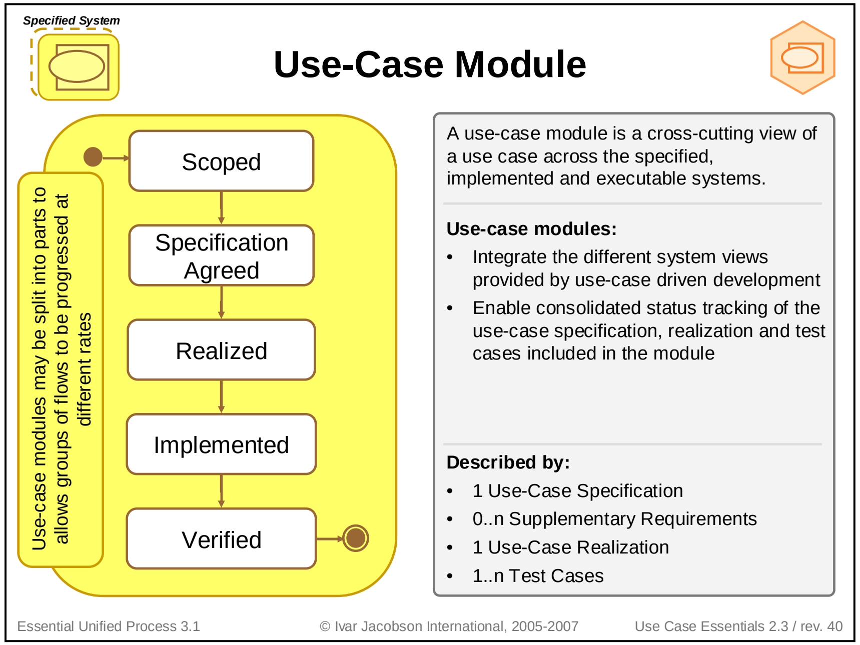


Inputs and expected results

- Insert email address with no '@'
- Verify that error message appears

Scenario derived from the Use Case

The Use-Case Module



Specified System

Use-Case Module

Scoped
Specification Agreed
Realized
Implemented
Verified
Use-case modules may be split into parts to allows groups of flows to be progressed at different rates

A use-case module is a cross-cutting view of a use case across the specified, implemented and executable systems.

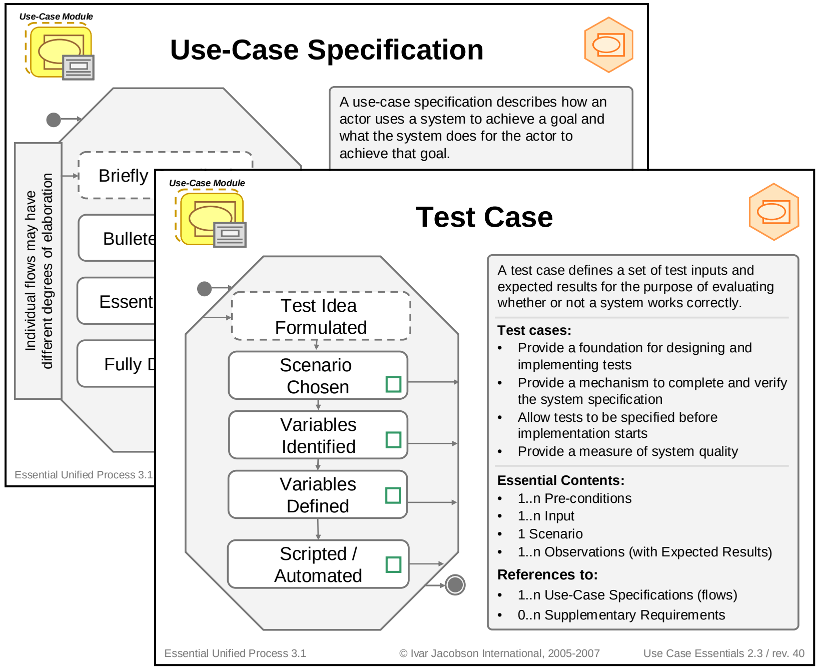
Use-case modules:

- Integrate the different system views provided by use-case driven development
- Enable consolidated status tracking of the use-case specification, realization and test cases included in the module

Described by:

- 1 Use-Case Specification
- 0..n Supplementary Requirements
- 1 Use-Case Realization
- 1..n Test Cases

Essential Unified Process 3.1 © Ivar Jacobson International, 2005-2007 Use Case Essentials 2.3 / rev. 40



Use-Case Module

Use-Case Specification

A use-case specification describes how an actor uses a system to achieve a goal and what the system does for the actor to achieve that goal.

Essential Unified Process 3.1

Use-Case Module

Test Case

A test case defines a set of test inputs and expected results for the purpose of evaluating whether or not a system works correctly.

Test cases:

- Provide a foundation for designing and implementing tests
- Provide a mechanism to complete and verify the system specification
- Allow tests to be specified before implementation starts
- Provide a measure of system quality

Essential Contents:

- 1..n Pre-conditions
- 1..n Input
- 1 Scenario
- 1..n Observations (with Expected Results)

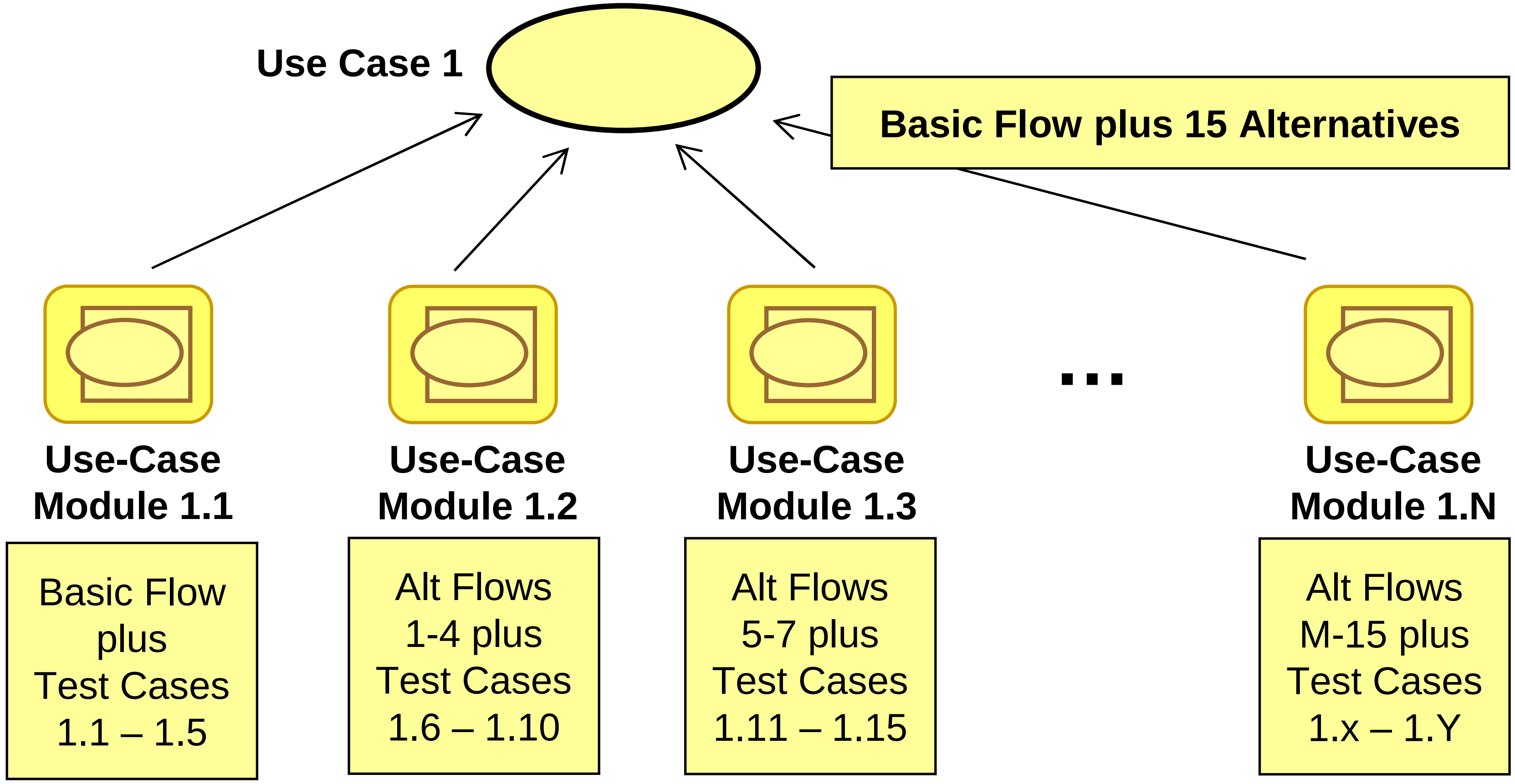
References to:

- 1..n Use-Case Specifications (flows)
- 0..n Supplementary Requirements

Essential Unified Process 3.1 © Ivar Jacobson International, 2005-2007 Use Case Essentials 2.3 / rev. 40

Use-Case Modules -
bringing Use Cases and Test Cases together to define done.

Use Cases and Use-Case Modules



Use Case 1

Basic Flow plus 15 Alternatives

Use-Case Module 1.1

Basic Flow plus Test Cases 1.1 – 1.5

Use-Case Module 1.2

Alt Flows 1-4 plus Test Cases 1.6 – 1.10

Use-Case Module 1.3

Alt Flows 5-7 plus Test Cases 1.11 – 1.15

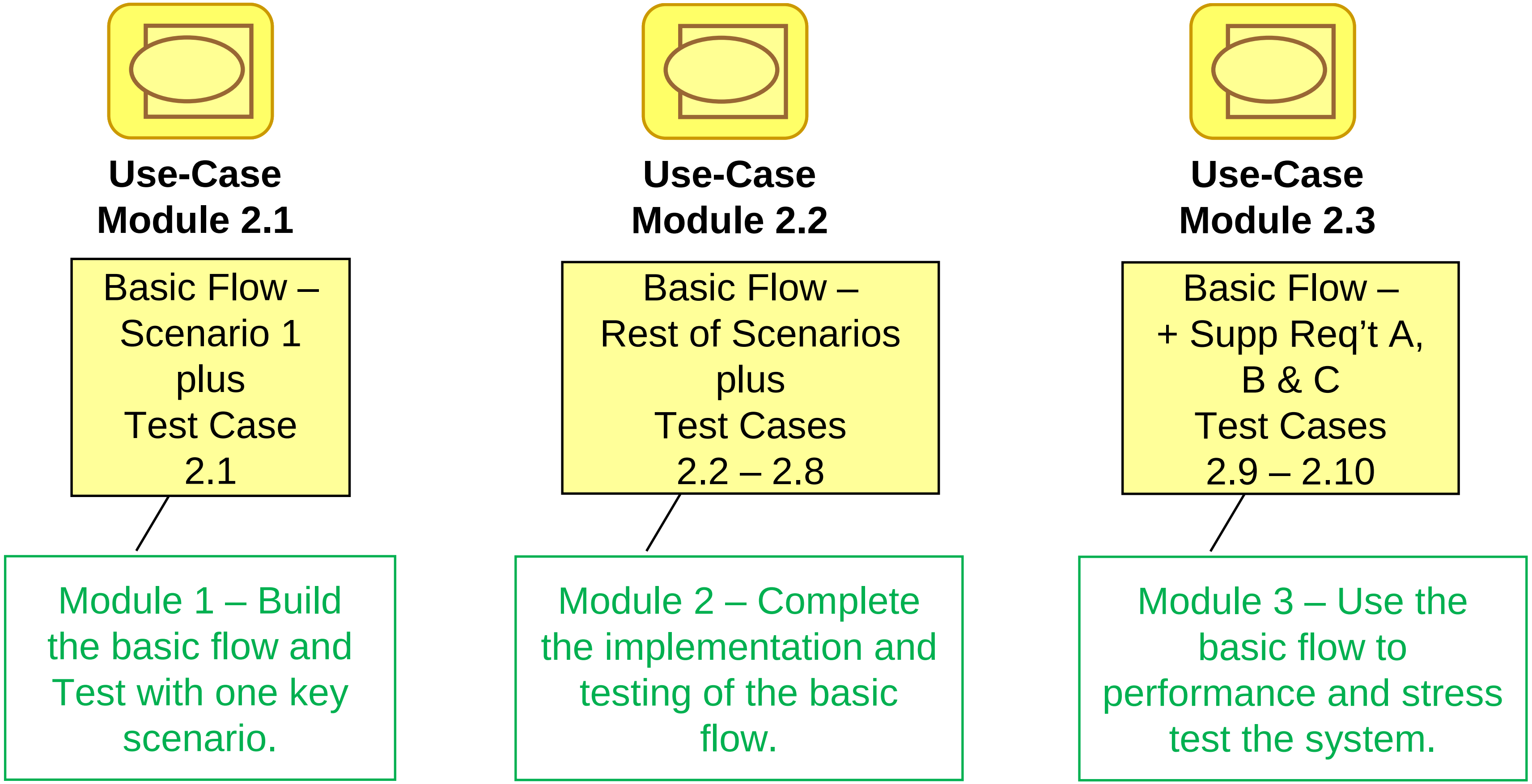
…

Use-Case Module 1.N

Alt Flows M-15 plus Test Cases 1.x – 1.Y

The Use-Case Modules split the use case up into a number of smaller, separately deliverable parts

Handling non-functional and other requirements



Use-Case Module 2.1

Basic Flow – Scenario 1 plus Test Case 2.1

Module 1 – Build the basic flow and Test with one key scenario.

Use-Case Module 2.2

Basic Flow – Rest of Scenarios plus Test Cases 2.2 – 2.8

Module 2 – Complete the implementation and testing of the basic flow.

Use-Case Module 2.3

Basic Flow – + Supp Req't A, B & C Test Cases 2.9 – 2.10

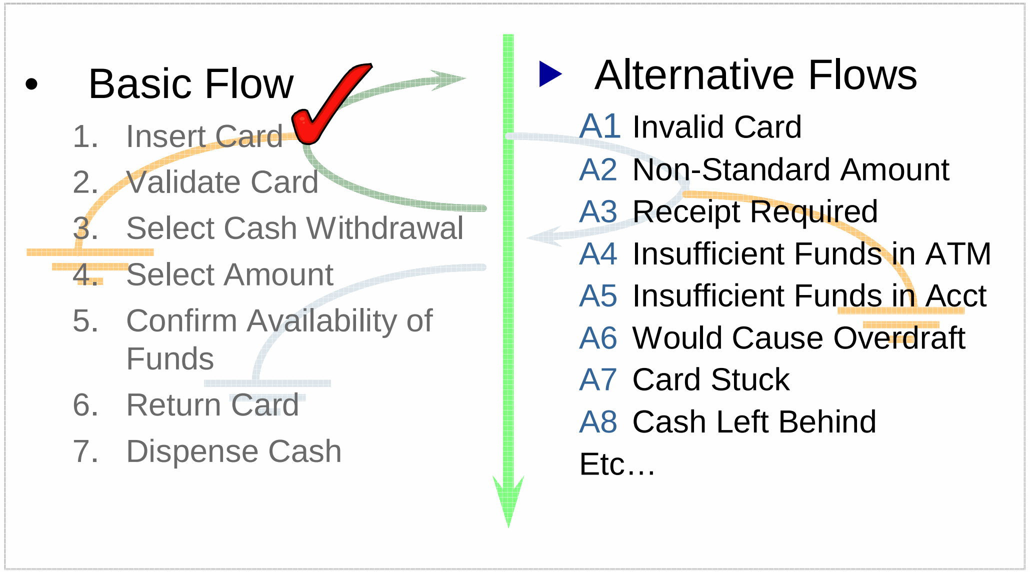
Module 3 – Use the basic flow to performance and stress test the system.

The modules can include test scenarios to address the non-functional as well as the functional requirements.

Finding Use-Modules

- **Think about your risks and identify the key scenarios**
- Think about the natural groups of flows
- Think about testing and proving the system
- Think work items and driving the development

Risks			
4	It might be harder than we think (estimates)	Very High	Build Withdraw Cash
5	Reliability of the O/S platform	Very High	Build Withdraw Cash
6	Scalability of J2EE Infrastructure	Very High	Build Withdraw Cash



- Basic Flow ✓
  1. Insert Card
  2. Validate Card
  3. Select Cash Withdrawal
  4. Select Amount
  5. Confirm Availability of Funds
  6. Return Card
  7. Dispense Cash

▶ Alternative Flows

A1 Invalid Card
A2 Non-Standard Amount
A3 Receipt Required
A4 Insufficient Funds in ATM
A5 Insufficient Funds in Acct
A6 Would Cause Overdraft
A7 Card Stuck
A8 Cash Left Behind
Etc…

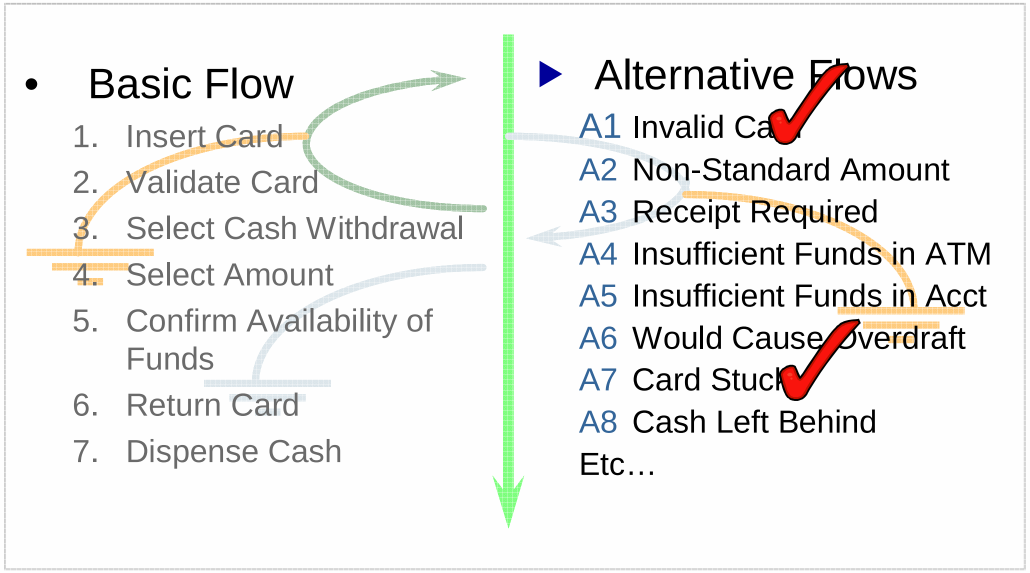
Build a simple cash withdrawal based on the Basic Flow

UCM 1.1 - Build a simple cash withdrawal based on the basic Flow

- One test case one account / one amount.

Finding Use-Modules

- Think about your risks and identify the key scenarios
- **Think about the natural groups of flows**
- Think about testing and proving the system
- Think work items and driving the development



- Basic Flow
  1. Insert Card
  2. Validate Card
  3. Select Cash Withdrawal
  4. Select Amount
  5. Confirm Availability of Funds
  6. Return Card
  7. Dispense Cash

▶ Alternative Flows

A1 Invalid Card ✓
A2 Non-Standard Amount
A3 Receipt Required
A4 Insufficient Funds in ATM
A5 Insufficient Funds in Acct
A6 Would Cause Overdraft
A7 Card Stuck ✓
A8 Cash Left Behind
Etc…

There are a number of flows about card handling?

Wouldn’t you implement them all at the same time?

UCM 1.2 – Card handling during cash withdrawal

A 1.1 Handle Invalid Card

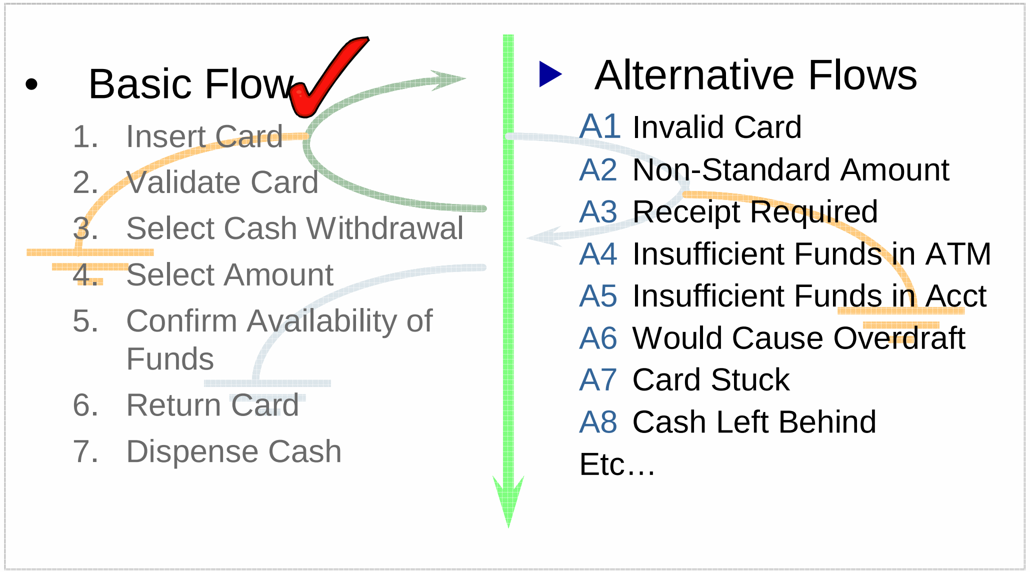
A 1.2 Handle Unreadable Card

A 1.3 Handle Card Jam

Numerous test cases

Finding Use-Modules

- Think about your risks and identify the key scenarios
- Think about the natural groups of flows
- **Think about testing and proving the system**
- Think work items and driving the development



- Basic Flow ✓
  1. Insert Card
  2. Validate Card
  3. Select Cash Withdrawal
  4. Select Amount
  5. Confirm Availability of Funds
  6. Return Card
  7. Dispense Cash

▶ Alternative Flows

A1 Invalid Card
A2 Non-Standard Amount
A3 Receipt Required
A4 Insufficient Funds in ATM
A5 Insufficient Funds in Acct
A6 Would Cause Overdraft
A7 Card Stuck
A8 Cash Left Behind
Etc…

How can we address the supplementary requirements?

How will we know when we’re done?

Performance 1.1: Peak Loading

Performance 1.2: Transaction Service Levels

UCM 1.3 – Peak Load Testing

Basic Flow

P 1.1 Peak Loading

P 1.2 Service Levels

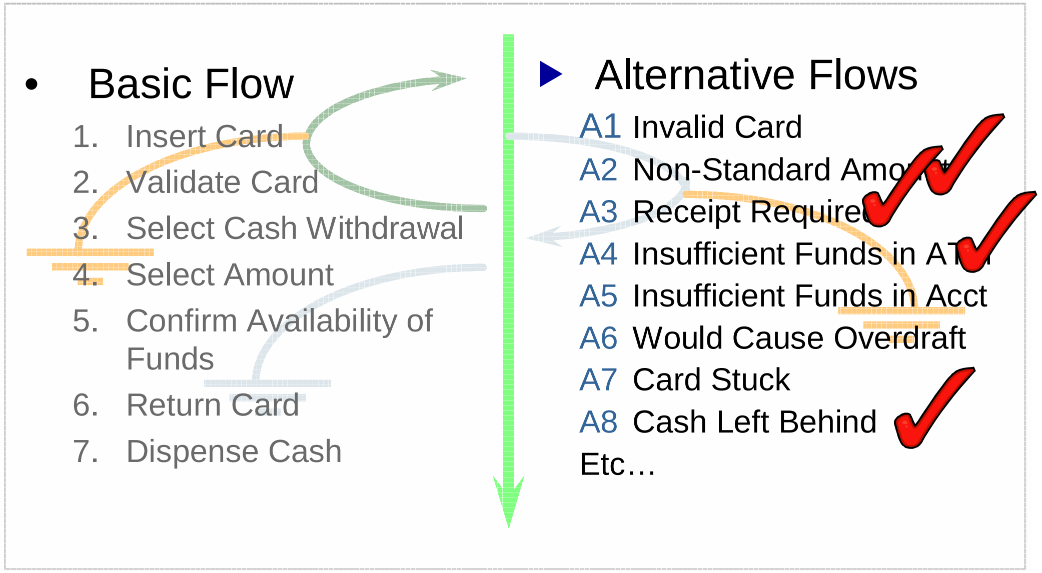
Numerous orchestrated test cases.

Finding Use-Modules

- Think about your risks and identify the key scenarios
- Think about the natural groups of flows
- Think about testing and proving the system
- **Think work items and driving the development**

What are we going to do in the next iteration – 2 weeks.

Well we can't do the whole use case?



- Basic Flow
  1. Insert Card
  2. Validate Card
  3. Select Cash Withdrawal
  4. Select Amount
  5. Confirm Availability of Funds
  6. Return Card
  7. Dispense Cash

- Alternative Flows
  - A1 Invalid Card
  - A2 Non-Standard Amount ✓
  - A3 Receipt Required ✓
  - A4 Insufficient Funds in ATM ✓
  - A5 Insufficient Funds in Acct
  - A6 Would Cause Overdraft
  - A7 Card Stuck
  - A8 Cash Left Behind ✓
  - Etc…

UCM 1.A – Handle Security Breaches

UCM 1.B – Handle Loss of Critical Resources

UCM 1.C – Forgetful Customer

UCM 1.D – Non-Standard Amounts

UCM 1.E – Receipt Handling

…..

Finding Use-Modules

- Think abc
- Think abc
- Think abc
- **Think wo**

- Basic
  1. Inser
  2. Valida
  3. Selec
  4. Selec
  5. Confi Fund
  6. Retur
  7. Dispe

What are we going to do in the
eks.

whole use

ecurity

oss of

Customer

dard

Handling

…..

Putting use-case modules onto cards

Use-Case Module 1.01

Manage reservations.

Use Case – 1 Cancel a reservation

Flows – A 2.1 Cancel reservation
A 2.2 Same day cancellation

On the front capture the name and flows covered by the module.

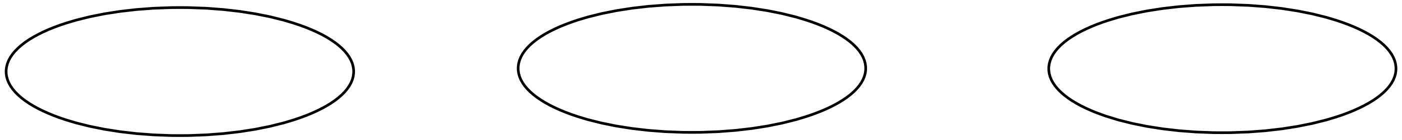
- □ **TC 1.01 Verify that a premium member can cancel the same day without a fee.**
- □ **TC 1.02 Verify that a non-premium member is charged 10% for a same day cancellation.**
- □ **TC 1.03 Verify that e-mail confirmations are sent.**
- □ **TC 1.04 Verify that the hotel is notified of any cancellation.**

On the back capture the test cases to be used as confirmation that the module is done.

Just like user stories the modules can be captured using index cards.

Sizing the work to be done

- Use cases can be any size



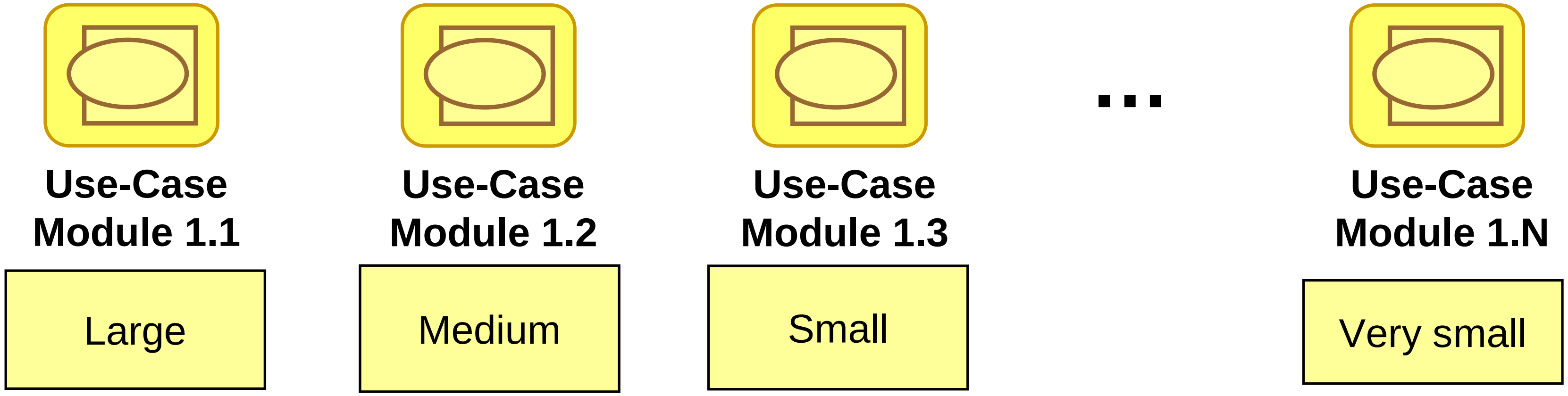
Use case 1
– 30 flows

Use case 2
– 3 flows

Use case 3
– 15 really long flows

  - And are often to big to describe, size, estimate or deliver in one go

- Use-case modules can be split up or combined to create sensibly sized work items



Use-Case Module 1.1

Large

Use-Case Module 1.2

Medium

Use-Case Module 1.3

Small

…

Use-Case Module 1.N

Very small

Estimating What You Can Do

Size	Trivial	Easy	Hard	Very Hard
Large	2	4	6	9
Medium	1	2	4	6
Small	0.5	1	2	4
Very Small	0.25	0.5	1	2

Complexity

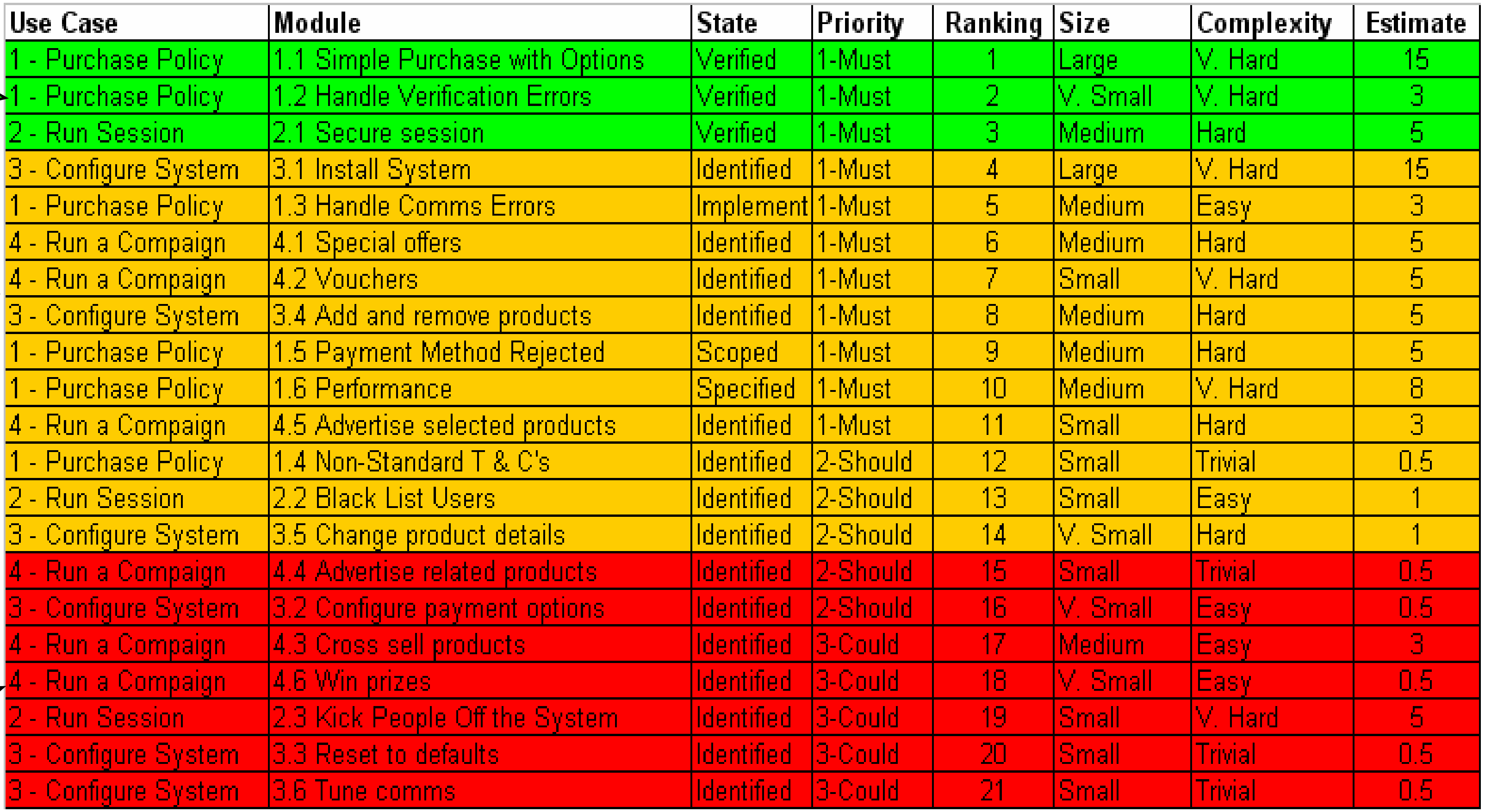
Building a backlog, tracking done

Done

Doable

Where time runs out.

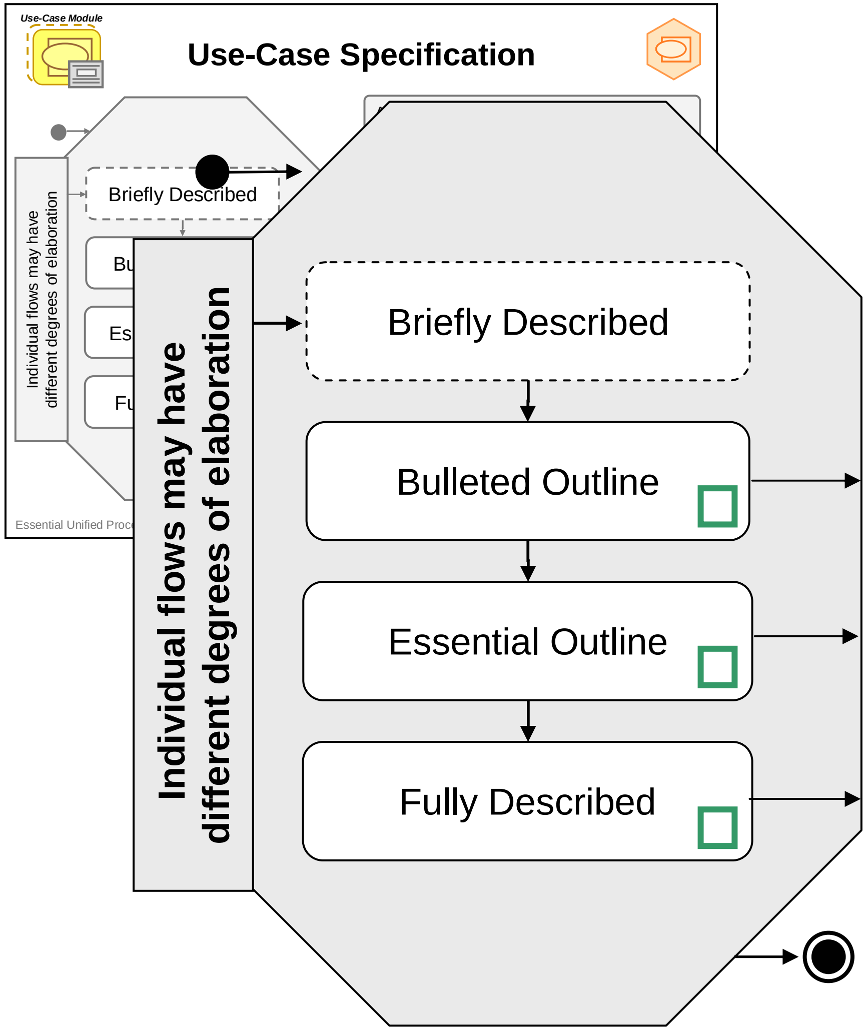
At risk

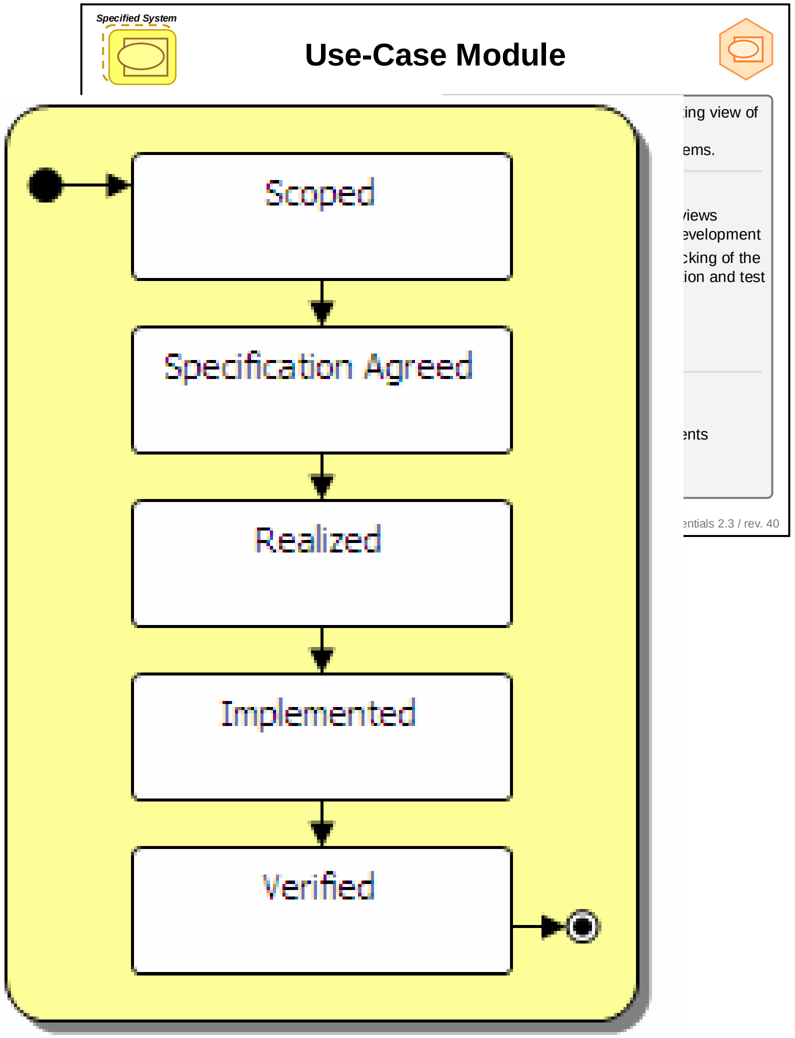


Use Case	Module	State	Priority	Ranking	Size	Complexity	Estimate
1 - Purchase Policy	1.1 Simple Purchase with Options	Verified	1-Must	1	Large	V. Hard	15
1 - Purchase Policy	1.2 Handle Verification Errors	Verified	1-Must	2	V. Small	V. Hard	3
2 - Run Session	2.1 Secure session	Verified	1-Must	3	Medium	Hard	5
3 - Configure System	3.1 Install System	Identified	1-Must	4	Large	V. Hard	15
1 - Purchase Policy	1.3 Handle Comms Errors	Implement	1-Must	5	Medium	Easy	3
4 - Run a Compaign	4.1 Special offers	Identified	1-Must	6	Medium	Hard	5
4 - Run a Compaign	4.2 Vouchers	Identified	1-Must	7	Small	V. Hard	5
3 - Configure System	3.4 Add and remove products	Identified	1-Must	8	Medium	Hard	5
1 - Purchase Policy	1.5 Payment Method Rejected	Scoped	1-Must	9	Medium	Hard	5
1 - Purchase Policy	1.6 Performance	Specified	1-Must	10	Medium	V. Hard	8
4 - Run a Compaign	4.5 Advertise selected products	Identified	1-Must	11	Small	Hard	3
1 - Purchase Policy	1.4 Non-Standard T & C's	Identified	2-Should	12	Small	Trivial	0.5
2 - Run Session	2.2 Black List Users	Identified	2-Should	13	Small	Easy	1
3 - Configure System	3.5 Change product details	Identified	2-Should	14	V. Small	Hard	1
4 - Run a Compaign	4.4 Advertise related products	Identified	2-Should	15	Small	Trivial	0.5
3 - Configure System	3.2 Configure payment options	Identified	2-Should	16	V. Small	Easy	0.5
4 - Run a Compaign	4.3 Cross sell products	Identified	3-Could	17	Medium	Easy	3
4 - Run a Compaign	4.6 Win prizes	Identified	3-Could	18	V. Small	Easy	0.5
2 - Run Session	2.3 Kick People Off the System	Identified	3-Could	19	Small	V. Hard	5
3 - Configure System	3.3 Reset to defaults	Identified	3-Could	20	Small	Trivial	0.5
3 - Configure System	3.6 Tune comms	Identified	3-Could	21	Small	Trivial	0.5

…and knowing how much more you can do.

Managing detail and driving the development





Agenda

- What are agile teams looking for?
- Being agile with use cases
- Case Studies
  - The agile sweet spot – a co-located team with an on-site customer
  - Scaling up the project – adding detail where it is needed
  - Outsourcing – working with external suppliers
- Demo
- Wrap Up / Getting Started

The agile sweet spot

- Small, co-located project team
- On-site customer / product owner
- Building a web-based insurance application
- 4 then 2 week iterations
- Previous experience of use cases
- No experience of iterating

Lightweight use cases to identify use-case modules.

Wrote test cases up front as they evolved their use cases.

First working software within four weeks.

Scaling up the project

- Large distributed project team
- Many stakeholders and sponsors
- Building a new banking straight through processing engine
- 6 then 4 and now 2 week iterations
- New to use cases
- New to agile and iteration

Started with more formal use cases and longer iterations.

Became more agile as they grew in confidence.

Delivered on-time and on-budget.

Outsourcing – working with external suppliers

- Outsourcing development to India
- Requirements and testing in the UK
- Building a retail banking application
- Working iteratively and incrementally
- Contractually need a formal requirements specification
- Distributed team – difficult to have timely conversations

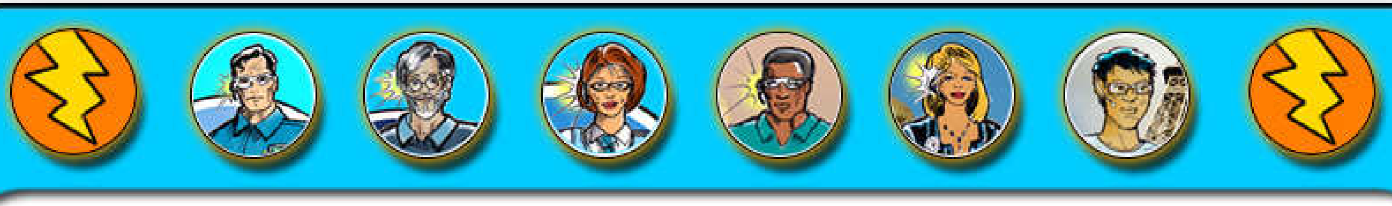
Use outlines and use-case modules to identify deliverable pieces of work.

Evolve the use cases to "fully described" to provide formal requirements specification.

Create test cases up front and use these to QA the releases delivered by the supplier.

Agenda

- What are agile teams looking for?
- Being agile with use cases
- Case Studies
- Demo
  - How can RequisitePro Help?
- Wrap Up / Getting Started



DEMO

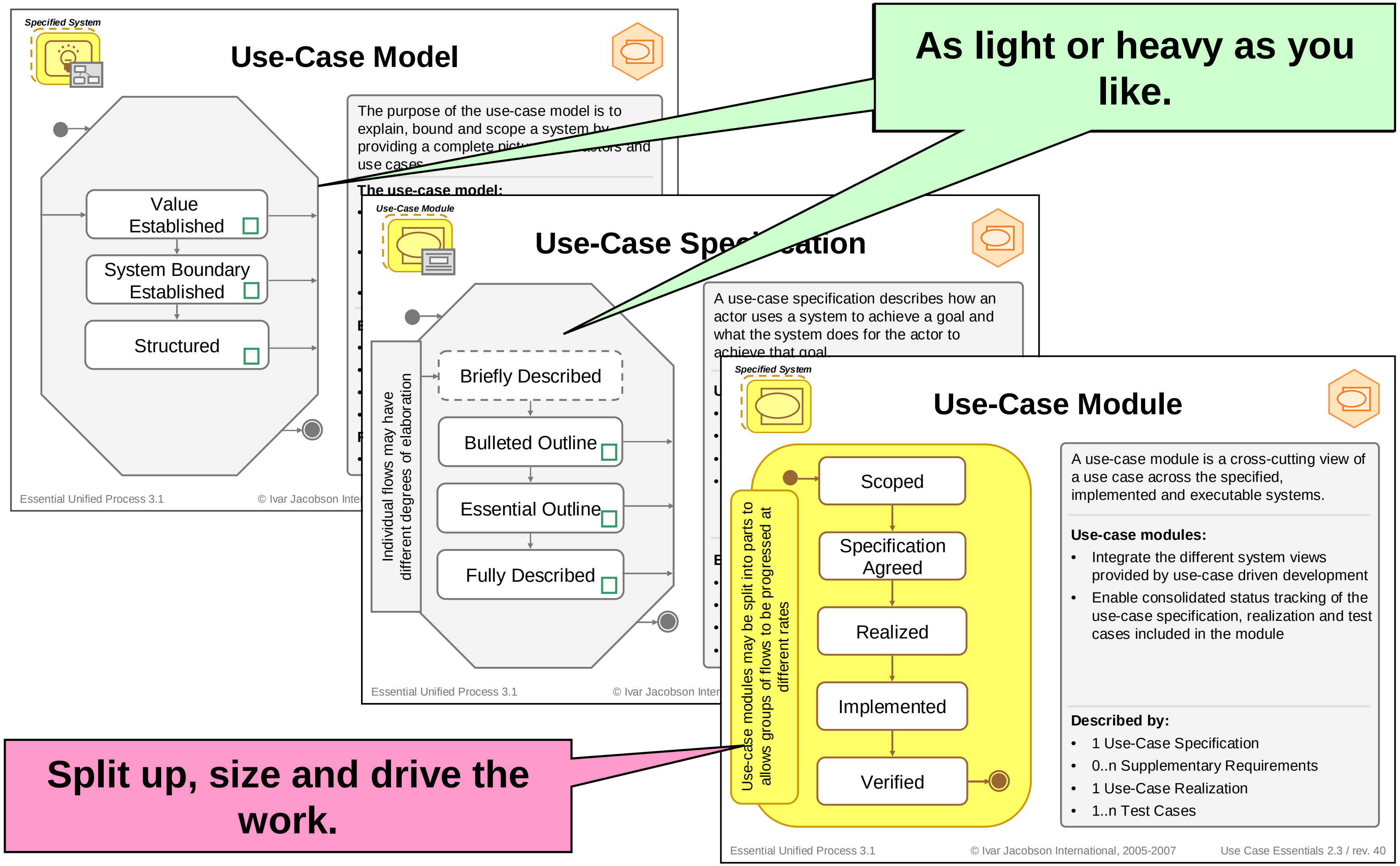
Agenda

- What are agile teams looking for?
- Being agile with use cases
- Case Studies
- Demo
- Wrap Up / Getting Started
  - Use cases or user stories: spot the difference
  - Being agile with use cases

Spot the Difference

	User Stories	Use Cases	Added Value
Quick and Lightweight	Stories on cards	Bulleted outlines	Can be evolved to add detail where necessary
	Placeholders for conversations	Placeholders for conversations	Added context for the conversations
Work items for the backlog	Stories are small (1 to 5 ideal days)	Modules are small (1 to 5 ideal days)	Epics, themes, stories and user types bought together into one easily understood model
	Story cards can be ripped up and replaced if too large	Use-case modules can be ripped up and replaced if too large	Nothing is lost as we still have the model and the original use cases
Definitions of done	Confirmation via test cases added to card	Test cases are an integral part of the module	The use case structure makes good test cases easy to find
	You never know when you've got all the stories	The model defines the whole system –easy to identify all the use cases and flows	The extent and scale of the system is readily apparent

Use Cases enable agility and scalability



Specified System

Use-Case Model

The purpose of the use-case model is to explain, bound and scope a system by providing a complete pictu actors and use cases

The use-case model:

Essential Unified Process 3.1 © Ivar Jacobson Inte

Use-Case Module

Use-Case Spec ation

A use-case specification describes how an actor uses a system to achieve a goal and what the system does for the actor to achieve that goal.

Essential Unified Process 3.1 © Ivar Jacobson Inter

Specified System

Use-Case Module

A use-case module is a cross-cutting view of a use case across the specified, implemented and executable systems.

Use-case modules:

- Integrate the different system views provided by use-case driven development
- Enable consolidated status tracking of the use-case specification, realization and test cases included in the module

Described by:

- 1 Use-Case Specification
- 0..n Supplementary Requirements
- 1 Use-Case Realization
- 1..n Test Cases

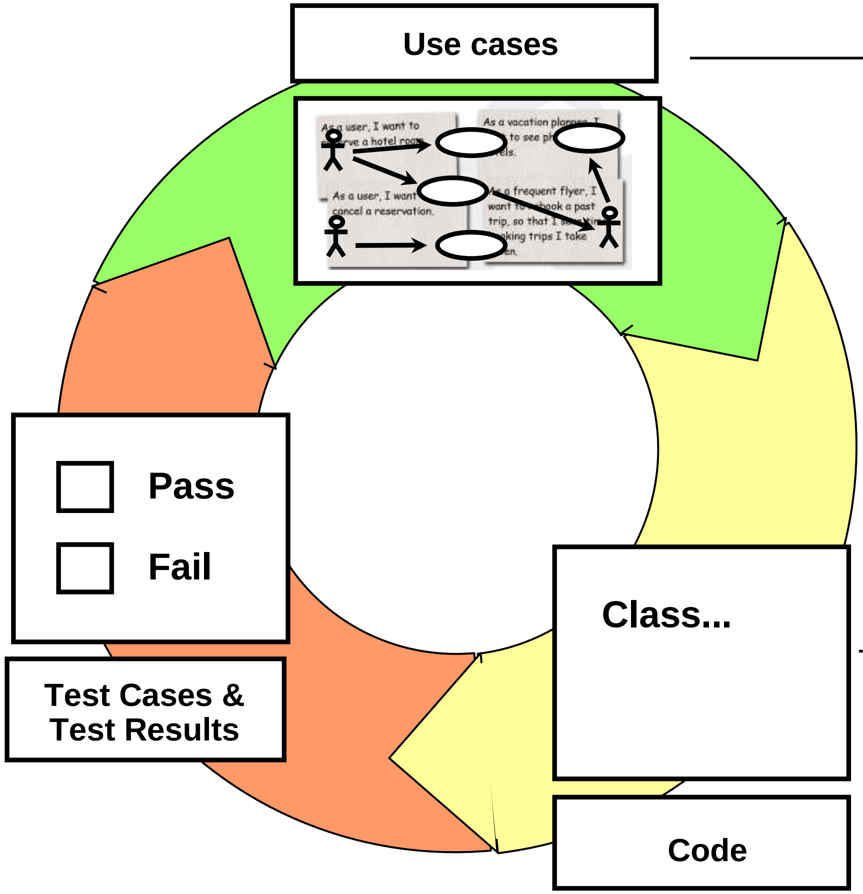
Use-case modules may be split into parts to allows groups of flows to be progressed at different rates
Scoped
Specification Agreed
Realized
Implemented
Verified

Essential Unified Process 3.1 © Ivar Jacobson International, 2005-2007 Use Case Essentials 2.3 / rev. 40

As light or heavy as you like.

Split up, size and drive the work.

Knowing what to do and when it’s done

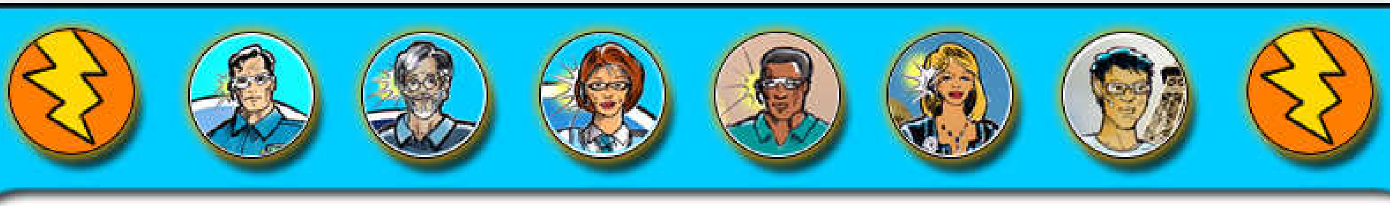

Use cases
As a user, I want to reserve a hotel room.
As a user, I want to cancel a reservation.
As a vacation planner, I want to see photos of the hotels.
As a frequent flyer, I want to rebook a past trip, so that I save time booking trips I take often.
Pass
Fail
Test Cases &
Test Results
Class...
Code

Lightweight use cases and module cards drive the work and act as placeholders for conversations to detail the requirements

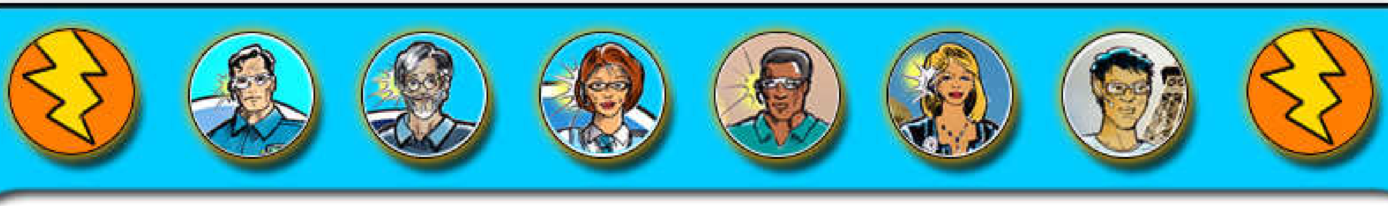
Tests verify that the system works well enough to satisfy the customers

Code implements the use-case modules

The benefits of an agile approach…
…with added context and scalability.



QUESTIONS



THANK YOU

Learn more at:

- IBM Rational software
- IBM Rational Software Delivery Platform
- Process and portfolio management
- Change and release management
- Quality management
- Architecture management

- Rational trial downloads
- Leading Innovation Web site
- developerWorks Rational
- IBM Rational TV
- IBM Rational Business Partners