Use Case—
How an Actor Interacts with the System

For such a simple and widely-used technique with common definitions, surprisingly little common understanding/usage and lots of misunderstanding and mis-usage
**Objectives**

- Describe common formats for representing use cases and issues they raise
- Distinguish types of requirements use cases do and do not show effectively
- Explain conventional use-case testing and how much it unknowingly misses

**Use Cases Typically Are Presumed to Be the User’s Requirements**

- Can be “Business Domain” use cases; but usually are “System” Use Cases describing *usage* of the (computer) system expected to be created [thus a form of high-level *design*—product/system/software requirements]
- Many use cases actually are GUI execution instructions, although most authorities recognize this as design and inappropriate for use cases
- Use cases often are treated with religious-like zeal which brooks no critical analysis

*User Stories → Use Cases ? Use Cases → User Stories*
**Requirements Overview**

- **Stakeholders**
  - Business needs, problems, value

- **Discovery Analysis**
  - REAL Business/ User/ Stakeholder Requirements

- **High-Level & Detailed**
  - DeliverableWhats → Value

- **Product/System/ Software Requirements**
  - Features Hows

- **Functional Requirements**
  - Use Cases (Usage)
  - Software Requirements Specifications

- **Non-Functional Requirements**
  - Quality Factors, Attributes, ‘lilities’
  - (Supplemental Specifications)

- **User Stories**

**Suggested Use Case Creation Steps**

1. Define the system boundaries
2. Identify the actors
3. Determine interactions
4. Establish pre & post conditions
5. Document the main success scenario
6. Branch to alternatives & exceptions
7. Merge or create sub-use cases as appropriate
8. Record additional information

Linda Westfall, “8 Steps to Effective Use Cases”
International Conference on Software Quality - ICSQ 09
ASQ Software Division November 2009 Northbrook, IL
**Actors Are Outside System Boundary--Users, Other Systems, Equipment**

- Who uses the system
- Who gets information from the system
- Who provides information to the system
- Who needs to be informed about certain occurrences within the system
- Who supports and maintains the system

*Primary Actor’s Goal provides a benefit, usually names the use case*

---

**Use Case Diagram**

[Diagram showing use cases and actors: Identify Customer, Verify Address, Order Items, Pay for Order, Bill Customer, Process Credit Card. Actors include Customer, Customer Service Rep, Credit Card Processor, and Billing System.]
Typical Use Case Textual Template

- Use case name
- Summary description
- Primary Actor
- Secondary Actors
- Trigger event that initiates use case
- Pre-conditions that must be present to start use case
- Post-conditions that must be present before use case can end successfully
- Priority, frequency of use, risk
- Scenario steps of primary Actor’s interaction with the system from primary Actor’s perspective
  - Main success flow (“Happy Path”)
  - Alternate success flows
  - Exception flows (don’t end with success)
- Special requirements

Exercise: Use Case Formats

One of the inconsistencies with use cases is that they are written with varying degrees and types of formality and elaboration in at least three different formats:

- Narrative text
- One-column
- Two-column

In your group, quickly review the following narrative text and determine whether it is a use case. How does it differ from a functional specification or technical design?
**Narrative Format: Identify Customer Use Case 1**

To place an order, a customer must be identified in the system. Once in the system, each customer has a unique Customer ID. When the Customer ID is entered, the specific customer’s record should be retrieved. Some customers have registered a credit card which they prefer to use. If the Customer ID is not known, the customer’s record can be located by entering the credit card number.

If the customer’s record cannot be found by exact match of Customer ID or credit card number, the customer’s record can be searched for alphabetically by the customer’s name. The name should be entered in last, first, middle name sequence. The name to be searched for (search argument) can be full or partial. The program will display a list of customers starting with the customer whose name is equal to or next greater than whatever has been entered as a search argument. There is no wild card logic. One can scroll backward and forward alphabetically through the list of names and addresses and select the record that is the customer’s.

**Narrative Format: Identify Customer Use Case 2**

If the customer’s record cannot be located, the customer may be added to the database and assigned a Customer ID consisting of the first three characters of the customer’s last name and the next sequential three-digit tiebreaker, starting with 000 for the first customer with those three characters. When adding a customer, the customer’s name, address, home/business phone numbers, and (optionally) a credit card number must be entered. Phone numbers should be 10 digits. The address should have a five- or nine-digit postal Zip code and a valid two-character state abbreviation.

Once the customer’s record has been retrieved/created and confirmed, go to the item entry routine.
Use cases are most commonly written in one-column format, from the primary Actor’s (e.g., clerk) perspective, such as:

1. **System displays menu choices.**
2. **Clerk selects Find by Customer ID**…
   or (less preferably) can embed system responses within Actor’s action steps,
3. **Clerk selects Find by Customer ID from the displayed menu choices**…

*In your group, quickly write the Happy Path (most common way use case is executed with success) from the narrative text as a series of numbered action steps.*

---

**Narrative Format: Identify Customer Use Case**

To place an order, a customer must be identified in the system. Once in the system, each customer has a unique Customer ID. When the Customer ID is entered, the specific customer’s record should be retrieved. Some customers have registered a credit card which they prefer to use. If the Customer ID is not known, the customer’s record can be located by entering the credit card number.

If the customer’s record cannot be found by exact match of Customer ID or credit card number, the customer’s record can be searched for alphabetically by the customer’s name. The name should be entered in last, first, middle name sequence. The name to be searched for (search argument) can be full or partial. The program will display a list of customers starting with the customer whose name is equal to or next greater than whatever has been entered as a search argument. There is no wild card logic. One can scroll backward and forward alphabetically through the list of names and addresses and select the record that is the customer’s.

If the customer’s record cannot be located, the customer may be added to the database and assigned a Customer ID consisting of the first three characters of the customer’s last name and the next sequential three-digit tiebreaker, starting with 000 for the first customer with those three characters. When adding a customer, the customer’s name, address, home/business phone numbers, and (optionally) a credit card number must be entered. Phone numbers should be 10 digits. The address should have a five- or nine-digit postal Zip code and a valid two-character state abbreviation.

Once the customer’s record has been retrieved/created and confirmed, go to the item entry routine.
After writing the Happy Path, each alternate success path (scenario) is written in the same format, but indicating where on the success path the alternate steps begin and where they return, for example:

A1.2.1 Clerk selects Find by Credit Card Number
A1.2.2 Clerk enters customer’s credit card number
A1.2.3 Return to 4

*In your group, quickly write one additional Alternate Success Path.*
Exercise: Alternate Success Paths

After writing the paths leading to successful completion, each exception path (scenario) that fails to successfully complete the use case is written in the same format, but indicating where on a success path the exception steps begin, for example:

E1.1 (A1.2.2) Clerk enters customer’s credit card number
E1.2 System displays “not found” message
E1.3 Clerk exits

How would you turn this Exception Failure Path into an Alternate Success Path?
Use Cases for Requirements and Testing--Facts and Follies

Exercise: Failure to Success Paths

E1.1 (A1.2.2) Clerk enters customer’s credit card number
E1.2 System displays “not found” message
A3.1 Clerk enters customer’s correct credit card number
A3.2 Return to 4

Issues with this (probably most common) format that lists each Alternate Success and Exception Failure Path separately?

Use Case

Strengths
- Step-by-step format is easy to understand
- Developers can develop systems that work the way users expect
- Readily translates into tests—execute each use case path

Weaknesses
- Format is not content, usage view may mask REAL requirements
- Can be as is rather than should be model
- Often includes only things expected to be automated
- Often misses paths other than Happy Path
- May actually create complexity
- Design rather than requirements
**Exercise: One-Column Use Cases**

1. System displays menu choices.
2. Clerk selects Find by Customer ID
3. Clerk enters customer’s ID
4. System retrieves and displays customer’s information
5. Clerk confirms displayed customer is correct
6. System goes to item entry routine

*User instructions for a presumed GUI design*

---

**Use Cases Are Not Well-Suited for:**

- Business rules
- Algorithms, formulas, and calculations
- Quality factors (non-functional requirements), including usability, performance, and security
- Configuration issues
- States, timing, and concurrency
- Data base contents and structure

*Splitting off into a separate supplemental specification interferes with understandability*
How Much to Write in Use Case

- Often extensive verbiage describing each step
- Separate alternate and exception scenarios add to the total verbiage
- Defeats advantage of being easily understood

Consider a full-view format logically depicting the flow choices rather than separately listing each Path--as a series of numbered succinct one-line Actor action and System response steps

Functionality Matrix

<table>
<thead>
<tr>
<th>User View (Use Cases)</th>
<th>Create</th>
<th>Retrieve</th>
<th>Update</th>
<th>Delete</th>
<th>Commun.</th>
<th>Interface</th>
<th>Logic</th>
<th>ChgState</th>
<th>PerfLevel</th>
<th>Constraint</th>
</tr>
</thead>
</table>
**Functionality Matrix: Identifies & Can Test Requirements Use Cases Miss**

User view, step-by-step
Technical view, what's happening “under the covers”
- **Create**, add, insert a new record in a file
- **Retrieve**, read, query an existing record in a file
- **Update**, modify, change an existing record in a file
- **Delete**, remove, scratch an existing record in a file
- **Communicate** with an external device
- **Interface** to another piece of software
- Perform **logic** or calculations
- **Change state**
- Meet a specified **performance** level
- Comply with an external **constraint**

Each user/technical view intersection should be addressed in a Test Design Specification (can split or consolidate)

---

**Mapping Logic Path**

*Not Just for Program Code*

- **Flowgraph: Node**
  - Module’s Entry, Exit
  - Where logic branches
  - Junctions (logic returns)
- **Flowgraph: Edge**
  - all logic between nodes
**Two-Column Use Cases**

**Look Even More Like Test Cases**

Left column is Actor/User action. Right column is System Response. Multiple possible responses are indicated, and flow to next step can be described. Numbering can be sequential or differentiated by Action/Response as shown.

*In your group, quickly flowgraph and identify paths to test*

<table>
<thead>
<tr>
<th>U1. Enter customer ID</th>
<th>R1.1. Customer is found (U4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1.2. Customer is not found (U2)</td>
</tr>
<tr>
<td>U2. Enter customer name</td>
<td>R2.1. Select customer from list (U4)</td>
</tr>
<tr>
<td></td>
<td>R2.2. Customer is not in list (U3)</td>
</tr>
<tr>
<td>U3. Add customer</td>
<td>R3. Customer is added (U3)</td>
</tr>
<tr>
<td>U4. Enter order</td>
<td>R4. Order is entered (Exit)</td>
</tr>
</tbody>
</table>

---

**Exercise: Flowgraph the Paths (Tests)**

[Flowgraph template]
Exercise: Other Test Conditions

In your group, write all the different inputs/conditions (in words, not with data values) that cause a specified use case path to be executed (instructor will assign a path to each group). Note: taken together all these inputs/conditions need to be demonstrated to give confidence that the specified use case path works.

What does this tell us about the premise that you need only a single test case for each use case scenario (path)?
Objectives

- Describe common formats for representing use cases and issues they raise
- Distinguish types of requirements use cases do and do not show effectively
- Explain conventional use-case testing and how much it unknowingly misses
President of Go Pro Management, Inc. consultancy since 1982, working directly with and training professionals in business engineering, requirements analysis, software acquisition, project management, quality and testing.

Partner with ProveIT.net in REAL ROI™ and ROI Value Modeling™.

Previously a developer, systems programmer/DBA/QA, and project leader with the City of Cleveland, leading financial institutions, and a “Big 4” consulting firm.

Degrees: Kenyon College, A.B.; Pennsylvania State University, M.S. in Psychology; Suffolk University, J.D.; Boston University, LL.M. in Tax Law.

Published author and frequent speaker at leading professional conferences.

Formerly International Vice President of the Association for Systems Management and Executive Editor of the Journal of Systems Management

Founding Chairman of the New England Center for Organizational Effectiveness.

Member of the Boston SPIN and SEPG’95 Planning and Program Committees.

Attendee Networking Coordinator for STAR, Better Software, and Test Automation Conferences.

Chair of record-setting attendance BOSCON 2000 and 2001, ASQ Boston Section’s Annual Quality Conferences.

Member IEEE Std. 829-2008 for Software Test Documentation Standard Revision Committee.

Member IEEE Std. 730-2014 standard for Software Quality Assurance Revision Committee.

International Institute of Business Analysis (IIBA) Business Analysis Body of Knowledge (BABOK) subject expert.

TechTarget SearchSoftwareQuality.com requirements and testing expert.

Admitted to the Massachusetts Bar and licensed to practice law in Massachusetts.

Author of books: Discovering REAL Business Requirements for Software Project Success

Author of forthcoming book: Cut Creep—Put Business Back in Business Analysis to Discover REAL Business Requirements for Agile, ATDD, and Other Project Success

Email me for updated slides
robin@gopromanagement.com www.gopromanagment.com