Use Cases:
The Good, The Bad, and The Ugly
(and what you can do about it)

Hadar Ziv
ziv@ics.uci.edu

In cooperation with:
Debra Richardson
Thomas Alspaugh
Thomas Standish
And the ROSATEA group at UCI
Presentation Outline

- **Use cases are good**
  - Quantum leap in software requirements specification (in principle)
- **Use cases are bad**
  - Difficult, time-consuming, and error-prone (in practice)
- **Use cases can get ugly**
  - Use case mistakes, misuse, and even “abuse cases”
- **What you can do about it**
  - Review “Top Ten” lists (practical advice)
  - Consider *Goals, Scenarios, Episodes, Concerns*, and *Aspects*
    (research work in progress)
Use Cases: The Good

Use cases are a simple and powerful way to define requirements for software behavior.
The Use-Case Model

- **A use-case model** illustrates
  - The system’s intended functions (use cases)
  - Its surroundings (actors)
  - Relationships between use cases and actors (use case diagram)
- The same use-case model used in requirements
  - Is used in analysis, design, and test
  - Serves as a unifying thread throughout system development

The most important role of a use-case model is to communicate the system’s functionality and behavior to the customer or end user
A Simple ATM Use Case Model

Withdraw Money
Deposit Money
Transfer Money (Between Accounts)

Bank Customer
ATM
Bank System
Use Case Details

- A use case is a textual or graphical description of
  - Major functions the system will perform for its actors
  - **Goals** the system achieves for its actors along the way

- A use case description should contain
  - Use case name
  - Basic course or path of action
  - Alternative paths and error/exception conditions

- **Scenarios**
  - Describe typical uses of the system as narrative
  - Correspond to a single path or flow through a use case
  - A use case is an abstraction or container of a set of related scenarios
Use Cases: The Bad and The Ugly

- If you don’t fully understand the ins and outs of use cases
  - It is easy to misuse them or turn them into “abuse cases”
- Ellen Gottesdiener
- Martin Fowler
- Doug Rosenberg
- Susan Lilly
- Kulak and Guiney
Ten Misguided Guidelines (Gottesdiener)

- Don’t bother with any other requirements representations
  - Use cases are the only requirements model you’ll need!
- Stump readers about the goal of your use case
  - Name use cases obtusely using vague verbs such as do or process
- Be ambiguous about the scope of your use cases
  - There will be scope creep anyway, so you can refactor your use cases later
- Include nonfunctional requirements and UI details in your use-case text
- Use lots of extends and includes in your initial use-case diagrams
  - This allows you to decompose use cases into itty bitty units of work
Ten Misguided Guidelines (Cont’d)

- Don’t be concerned with defining business rules
  – you’ll probably remember some of them when you design and code
- Don’t involve subject matter experts in creating, reviewing, or verifying use cases
  – They’ll only raise questions!
- If you involve users at all in use case definition, just “do it”
  – Why bother to prepare for meetings with the users?
- Write your first and only use case draft in excruciating detail
  – Why bother iterating with end users when they don’t even know what they want
- Don’t validate or verify your use cases
  – That will only cause you to make revisions and do more rework!
Top Use Case Mistakes (Rosenberg)

- Don’t write functional requirements instead of usage scenario text
  - Requirements are generally stated in terms of what the system shall do
  - Usage scenarios are user actions and corresponding system responses
- Don’t describe attributes and methods rather than usage
  - Don’t include too many presentation details
  - Don’t detail data-entry fields on user screen
- Don’t write the use cases too tersely
  - Must describe user actions and system responses in detail
  - Err on the side of too much detail in user documentation
- Don’t completely ignore the user interface
  - Discuss features that allow the user to tell the system to “do something”
- Don’t avoid explicit names for boundary objects
  - Name boundary objects explicitly in the use case text
Top Use Case Mistakes (Cont’d)

- **Don’t write in a passive or not the user’s voice**
  - Should be written from the user’s perspective
  - Present-tense verb phrases in active voice

- **Don’t ignore system behavior**
  - Include what the system does in response to user actions
    » Creates new objects
    » Validates user input
    » Generates error messages

- **Don’t omit text for alternative courses of action**
  - Basic course of action easier to identify and write
  - But alternate courses are critical for correctness and completeness; robustness

- **Don’t focus on things outside the use case**
  - Such as how you get there or what happens afterwards
  - Watch out for “long form” use case templates!

- **Don’t spend a month deciding whether to use includes or extends**
What Can Be Done About It?

- Question everything, even the basic definitions of relationships between
  - Use cases to **Goals** (1:1?)
  - Use cases to **Scenarios** (1:m?)
  - **Goals** to **Scenarios**?
  - All of the above to design and implementation???
Use Cases and Goals

- Use cases correspond to **goals**
  - A *goal* is a “desired state of affairs” (Schank/Wilensky)

- **Goals** have nontrivial structure and relationships
  - At least hierarchical but could be more complex
  - *We need better understanding and analysis of goals*

- ATM example
  - High-level stakeholder *goals*
    » Increase the bank’s business success
    » Increase market share
    » Provide greater access to banking services
  - Low-level *goals*
    » Terminate a user’s session
    » Authenticate a user’s ATM card and PIN
    » Withdraw $200 cash from user’s account
(Partial) Requirements Goal Graph

increase Bank’s business success

increase profitability

increase productivity

invest in automation

automate human teller banking services using Automated Teller Machines (or ATMs)

minimize total system development and deployment costs

increase market share

increase customer satisfaction

provide greater access to banking services

more available hours

more available locations

safe and secure

convenient and easy to use

Hadar Ziv / ziv@ics.uci.edu
Use Cases and Scenarios

- Use cases contain a family of related *scenarios*
  - Within a single use case, *scenarios* may have nontrivial structure
  - Across use cases, *scenarios* are often referred to, reused, or linked in nontrivial ways
  - Often, containment becomes confinement!

- **Scenarios**
  - A sequence of events that corresponds to a purposeful use of a system
  - “Purposeful uses” are characterized by associated *goals*

- **Episodes**
  - Subsequences of events contained within a surrounding *scenario*
  - Correspond to the pursuit of *subgoals*
  - Example *episodes*: Login, Logout, Authenticate
Goals and Scenarios

- Software requirements
  - Involve many goals at many different levels of abstraction/detail
    » “provide” goals
    » “prevent” goals

- Goals and Scenarios
  - A high-level “provide” goal typically corresponds to a single usage scenario
  - A low-level “provide” or “prevent” goal typically corresponds to a single plan of action or “episode”
  - Therefore, a usage scenario consists of multiple episodes addressing multiple goals!
A Family of ATM Scenarios

1. Insert ATM Card
2. Give PIN to Access Your Account
3. Choose a Transaction
   - Withdraw Cash
   - Make Deposit
   - Find Balance
   - Log Off
4. ATM Machine Returns Your ATM Card
A Family of ATM Scenarios (Cont’d)

1. "Login Episode"
2. The ATM presents, in English, a choice of transactions the customer may perform.
3. Iteration *:
   1. Alternatives:
      1. Alternative:
         1. The customer selects "Withdraw cash".
         2. "Withdraw Cash Episode"
      2. Alternative:
         1. The customer selects "Make deposit".
         2. "Deposit Funds Episode"
      3. Alternative:
         Guard: Customer has more than one account.
         1. The customer selects "Transfer funds".
         2. "Transfer Funds Episode"
      4. Alternative:
         1. The customer selects "Balance".
         2. "Balance Episode"
      2. ATM presents, in French, a choice of transactions the customer may perform.
        4. The customer selects "Done".
      5. The ATM ejects the ATM card and beeps until the customer withdraw it.
      6. The customer withdraws the card.
A Sample Login Episode

1. The **customer** inserts an **ATM card** into an **ATM**.
2. The **ATM** presents a **choice of languages**.
3. The **customer** selects **English**.
4. The **ATM** prompts for a **PIN**.
5. The **customer** enters the **PIN** for his/her **ATM** card.
A Sample Withdraw Cash Episode

1. The **customer** selects "Withdraw cash".
2. The **ATM** presents the **accounts** from which the **customer** can withdraw.
3. The **customer** selects "Checking".
4. The **ATM** prompts for an amount to withdraw.
5. The **customer** enters $200.
6. The **ATM** dispenses the requested amount of cash.
7. The **ATM** prints a receipt.
8. The **ATM** presents a choice of transactions the **customer** may perform.
Example of Scenario Goal Analysis:

- **g2**: Use ATM to withdraw $200 cash from your checking account
  - **g2a**: Get access to ATM banking services in your language
    - **g2a1**: Insert ATM card
      - {1}
    - **g2a2**: Select French as UI language
      - {2,3}
    - **g2a3**: Enter PIN for ATM card
      - {4,5}
  - **g2b**: Use ATM’s Cash Withdrawal service to withdraw $200 in cash
    - **g2b1**: Choose transaction “Withdraw Cash”
      - {6,7}
    - **g2b2**: Designate “Checking Account” to use for withdrawal
      - {8,9}
    - **g2b3**: Specify $200 as amount to withdraw
      - {10,11}
  - **g2c**: Finish ATM session properly
    - **g2c1**: Choose transaction “Done”
      - {14,15}
    - **g2c2**: Collect ATM card from machine
      - {16,17}
ATM AND/OR Goal Analysis

Access and Authorization Goals

- grant access to qualified users
- deny access to unqualified users

Allow users who have been granted access to perform authorized transactions

- make deposit D
- find current account balance B
- withdraw cash amount W

credit deposit D to balance B
B ← B + D,
tell customer when B is available for withdrawal

W ≤ Balance B
W ≤ Bank’s daily withdrawal limit L
W ≤ ATM’s cash C available to dispense

withdrawal permitted if
AND

W ≤ Bank’s daily withdrawal limit L
W ≤ ATM’s cash C available to dispense

Debit withdrawal from balance B
B ← B - W
Concerns and Aspects

- Goals correspond to concerns
  - “provide” concerns
    » Withdraw money, Deposit money, Transfer money
  - “prevent” concerns
- Aspects correspond to cross-cutting concerns
  - Typically “prevent” concerns
    » User access/authentication, data integrity, transaction integrity
- A requirements-level usage-scenario
  - Will be written as a collection of episodes
  - Will be designed to address multiple concerns
  - Will be implemented using “regular” code + aspects for the cross cutting concerns
Use Cases and Aspects

- According to Jacobson,
  - All use cases are extensions to the “null system”
- He sees a relationship between use cases and aspects, such that
  - aspects ≈ extensions
  - join points ≈ extension points
- AOP allows us to
  - Separate use case extensions all the way down to code
  - Compose back extensions before execution
- Thus, AOP supports extensions
  - Ivar Jacobson, “Use cases and aspects – Working together.”
Summary and Recommendations

- Be aware of “top ten” lists of use case mistakes, misuse, and “abuse cases”
  - Beware each article has a different list!😊
  - Beware articles provide different, sometimes conflicting advice!😊

- Consider goals and scenarios
  - When writing use cases or instead of writing them
  - Perform goal analysis and goal decomposition
  - Perform scenario analysis and scenario composition (from episodes)
  - Design and implement using concerns and aspects (for cross-cutting concerns)