

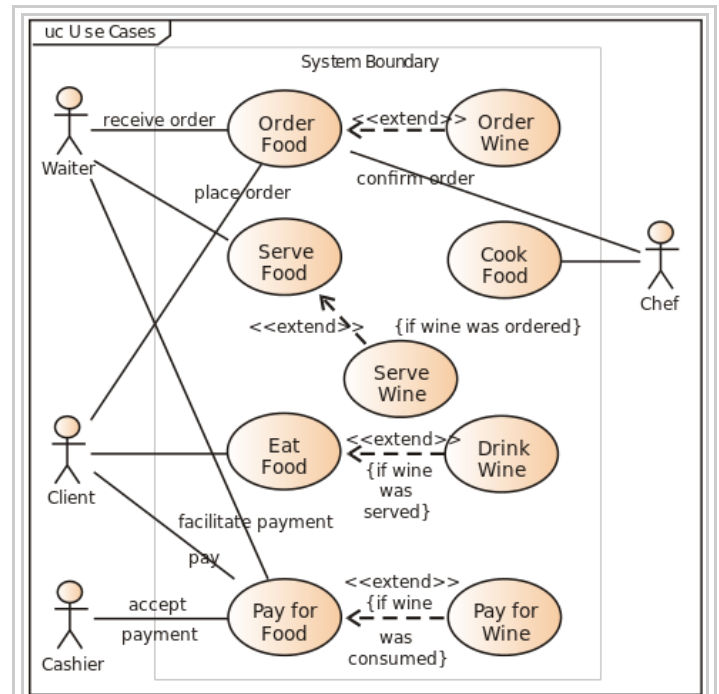
Use case

From Wikipedia, the free encyclopedia

In software and systems engineering, a **use case** is a list of steps, typically defining interactions between a role (known in Unified Modeling Language (UML) as an "actor") and a system, to achieve a goal. The actor can be a human, an external system, or time.

In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in Systems Modeling Language (SysML) or as contractual statements.

As an important requirement technique, use cases have been widely used in modern software engineering over the last two decades. Use case driven development is a key characteristic of process models and frameworks like Unified Process (UP), Rational Unified Process (RUP), Oracle Unified Method (OUM), etc. With its iterative and evolutionary nature, use case is also a good fit for agile development.



A UML Use Case Diagram for the interaction of a client (the actor) and a restaurant (the system)

Contents

- 1 History
- 2 Templates
 - 2.1 Martin Fowler
 - 2.2 Alistair Cockburn
 - 2.2.1 Fully dressed
 - 2.2.2 Casual
 - 2.2.3 Design scopes
 - 2.2.4 Goal levels
- 3 Actors
- 4 Visual Modeling
- 5 Example
- 6 Advantages
- 7 Limitations

- 8 Misconceptions
- 9 Tools
- 10 See also
- 11 References
- 12 Further reading
- 13 External links

History

In 1986 Ivar Jacobson first formulated textual, structural, and visual modeling techniques for specifying use cases. In 1992 his co-authored book *Object-Oriented Software Engineering - A Use Case Driven Approach*^[1] helped to popularize the technique for capturing functional requirements, especially in software development. Originally he had used the terms *usage scenarios* and *usage case* — the latter a direct translation of his Swedish term *användningsfall* — but found that neither of these terms sounded natural in English, and eventually he settled on *use case*.^[2] Since then, others have contributed to improving this technique, notably Alistair Cockburn, Larry Constantine, Dean Leffingwell, Kurt Bittner and Gunnar Overgaard.

In 2011 Jacobson published an update to his work, called *Use Case 2.0*,^[3] with the intention of incorporating many of his practical experiences of applying use cases since the original inception of the concept.^[4]

Templates

Martin Fowler

Martin Fowler states "There is no standard way to write the content of a use case, and different formats work well in different cases."^{[5]:100} He describes "a common style to use" as follows:^{[5]:101}

- Title: "goal the use case is trying to satisfy"^{[5]:101}
- Main Success Scenario: numbered list of steps^{[5]:101}
 - Step: "a simple statement of the interaction between the actor and a system"^{[5]:101}
- Extensions: separately numbered lists, one per Extension^{[5]:101}
 - Extension: "a condition that results in different interactions from .. the main success scenario".
An extension from main step 3 is numbered 3a, etc.^{[5]:101}

Alistair Cockburn

Alistair Cockburn describes a more detailed structure for a use case, but permits it to be simplified when less detail is needed. His "fully dressed" use case structure is:[6]:9 – 10

Fully dressed

Cockburn's fully dressed use case template lists the following fields:[7]

- Title: "an active-verb goal phrase that names the goal of the primary actor"[8]
- Primary Actor
- Goal in Context
- Scope
- Level
- Stakeholders and Interests
- Precondition
- Minimal Guarantees
- Success Guarantees
- Trigger
- Main Success Scenario
- Extensions
- Technology & Data Variations List
- Related Information.

In addition, Cockburn suggests using two devices to indicate the nature of each use case: icons for design scope and goal level.

Cockburn's approach has influenced other authors; for example, Alexander and Beus-Dukic generalize Cockburn's "Fully dressed use case" template from software to systems of all kinds, with the following fields differing from Cockburn:[9]

- Variation scenarios "(maybe branching off from and maybe returning to the main scenario)"
- Exceptions "i.e. exception events and their exception-handling scenarios"

Casual






Cockburn recognizes that projects may not always need detailed "fully dressed" use cases. He describes a Casual use case with the fields:[7]

- Title (goal)
- Primary Actor
- Scope
- Level

- (Story): the body of the use case is simply a paragraph or two of text, informally describing what happens.

Design scopes



Cockburn suggests annotating each use case with a symbol to show the "Design Scope", which may be black-box (internal detail is hidden) or white-box (internal detail is shown). Five symbols are available:^[10]

Scope	Icon	
Organization (black-box)	Filled House	
Organization (white-box)	Unfilled House	
System (black-box)	Filled Box	
System (white-box)	Unfilled Box	
Component	Screw or Bolt	

Other authors sometimes call use cases at Organization level "Business use cases".^[11]

Goal levels

Cockburn suggests annotating each use case with a symbol to show the "Goal Level";^[12] the preferred level is "User-goal" (or colloquially "sea level"^{[5]:101}).

Goal Level	Icon	Symbol	
Very High Summary	Cloud	++	
Summary	Flying Kite	+	
User Goal	Waves at Sea	!	
Subfunction	Fish	-	
Too Low	Seabed Clam-Shell	--	

Sometimes in text writing, a use-case name followed by an alternative text symbol (!, +, -, etc.) is a more concise and convenient way to denote levels, e.g. *place an order!*, *login-*.

Actors

A use case defines the interactions between external actors and the system under consideration to accomplish a goal. Actors must be able to make decisions, but need not be human: "An actor might be a person, a company or organization, a computer program, or a computer system—hardware, software, or both."^[13] Actors are always stakeholders, but not all stakeholders are actors, since they "never interact directly with the system, even though they have the right to care how the system behaves."^[13] For example, "the owners of the system, the company's board of directors, and regulatory bodies such as the Internal Revenue Service and the Department of Insurance" could all be stakeholders but are unlikely to be actors.^[13]

Similarly, a person using a system may be represented as different actors because he is playing different roles. For example, user "Joe" could be playing the role of a Customer when using an Automated Teller Machine to withdraw cash from his own account, or playing the role of a Bank Teller when using the system to restock the cash drawer on behalf of the bank.

Actors are often working on behalf of someone else. Cockburn writes that "These days I write 'sales rep for the customer' or 'clerk for the marketing department' to capture that the user of the system is acting for someone else." This tells the project that the "user interface and security clearances" should be designed for the sales rep and clerk, but that the customer and marketing department are the roles concerned about the results.^[14]

A stakeholder may play both an active and an inactive role: for example, a Consumer is both a "mass-market purchaser" (not interacting with the system) and a User (an actor, actively interacting with the purchased product).^[15] In turn, a User is both a "normal operator" (an actor using the system for its intended purpose) and a "functional beneficiary" (a stakeholder who benefits from the use of the system).^[15] For example, when user "Joe" withdraws cash from his account, he is operating the Automated Teller Machine and obtaining a result on his own behalf.

Cockburn advises to look for actors among the stakeholders of a system, the primary and supporting (secondary) actors of a use case, the system under design (SuD) itself, and finally among the "internal actors", namely the components of the system under design.^[13]

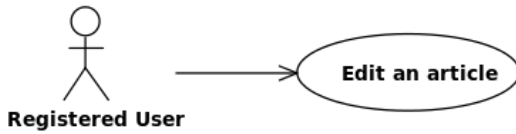
Visual Modeling

In the Unified Modeling Language, the relationships between all (or a set of) the use cases and actors are represented in a Use Case Diagram or diagrams, originally based upon Ivar Jacobson's Objectory notation. SysML, a UML profile, uses the same notation at the system block level.

Besides Use Case Diagram, behavioral UML diagrams like Activity diagram, Sequence diagram, Communication diagram and State machine diagram are often used to visualize a use case.

Example

Below is a sample use case written with a slightly-modified version of the Cockburn-style template. Note that there are no buttons, controls, forms, or any other UI elements and operations in the basic use case description, where only user goals, subgoals or intentions are expressed in every step of the basic flow or extensions. This practice makes the requirement specification clearer, and maximizes the flexibility of the design and implementations.



Use Case:

Edit an article

Primary Actor:

Member (*Registered User*)

Scope: a Wiki system

Level: ! (*User goal or sea level*)

Brief: (*equivalent to a user story or an epic*)

The member edits any part (the entire article or just a section) of an article he/she is reading. Preview and changes comparison are allowed during the editing.

Stakeholders

...

Postconditions

Minimal Guarantees:

Success Guarantees:

- The article is saved and an updated view is shown.
- An edit record for the article is created by the system, so watchers of the article can be informed of the update in a while later.

Preconditions:

The article with editing enabled is presented to the member.

Triggers:

The member invokes an edit request (for the full article or just one section) on the article.

Basic flow:

1. The system provides a new editor area/box filled with all the article's relevant content with an informative edit summary for the member to edit. If the member just wants to edit a section of the article, only the original content of the section is shown, with the section title automatically filled out in the edit summary.
2. The member modifies the article's content till satisfied.
3. The member fills out the edit summary, tells the system if he/she wants to watch this article, and submits the edit.
4. The system saves the article, logs the edit event and finishes any necessary post processing.
5. The system presents the updated view of the article to the member.

Extensions:

2-3.

a. Show preview:

1. The member selects *Show preview* which submits the modified content.
2. The system reruns step 1 with addition of the rendered updated content for preview, and informs the member that his/her edits have not been saved yet, then continues.

b. Show changes:

1. The member selects *Show changes* which submits the modified content.
2. The system reruns step 1 with addition of showing the results of comparing the differences between the current edits by the member and the most recent saved version of the article, then continues.

c. Cancel the edit:

1. The member selects *Cancel*.
2. The system discards any change the member has made, then goto step 5.

4a. Timeout:

...

Advantages

Alistair Cockburn lists 5 reasons why he still writes use cases in agile development.^[16]

1. The list of goal names provides the shortest summary of what the system will offer (even than user stories). It also provides a project planning skeleton, to be used to build initial priorities, estimates, team allocation and timing.
2. The main success scenario of each use case provides everyone involved with an agreement as to what the system will basically do and what it will not do. It provides the context for each specific line item requirement (e.g. fine-grained user stories), a context that is very hard to get anywhere else.
3. The extension conditions of each use case provide a framework for investigating all the little, niggling things that somehow take up 80% of the development time and budget. It provides a look ahead mechanism, so the stakeholders can spot issues that are likely to take a long time to get answers for. These issues can and should then be put ahead of the schedule, so that the answers can be ready when the development team gets around to working on them.
4. The use case extension scenario fragments provide answers to the many detailed, often tricky and ignored business questions: “What are we supposed to do in this case?” It is a thinking/documentation framework that matches the if...then...else statement that helps the programmers think through issues. Except it is done at investigation time, not programming time.
5. The full use case set shows that the investigators have thought through every user’s needs, every goal they have with respect to the system, and every business variant involved.

Limitations

Limitations of use cases include:

- Use cases are not well suited to capturing non-interaction based requirements of a system (such as algorithm or mathematical requirements) or non-functional requirements (such as platform, performance, timing, or safety-critical aspects). These are better specified declaratively elsewhere.
- Use case templates do not automatically ensure clarity. Clarity depends on the skill of the writer(s).
- For some products and systems, use cases are complex to write and to understand, for both end users and developers.
- As there are no fully standard definitions of use cases, each project must form its own interpretation.
- Some use case relationships, such as *extends*, are ambiguous in interpretation and can be difficult for stakeholders to understand.
- In Agile development, especially Extreme programming, simpler user stories are preferred to use cases.
- Use case developers often find it difficult to determine the level of user interface (UI) dependency to incorporate in a use case. While use case theory suggests that UI not be reflected in use cases, it can be awkward to abstract out this aspect of design, as it makes the use cases difficult to visualize. In

software engineering, this difficulty is resolved by applying requirements traceability, for example with a traceability matrix. Another approach to associate UI elements with use cases, is to attach a UI design to each step in the use case. This is called a use case storyboard.

- Use cases can be over-emphasized. Bertrand Meyer discusses issues such as driving system design too literally from use cases, and using use cases to the exclusion of other potentially valuable requirements analysis techniques.^[17]
- Use cases are a starting point for test design,^[18] but since each test needs its own success criteria, use cases may need to be modified to provide separate post-conditions for each path.^[19]

Misconceptions

- User story is agile, use case is not.

Agile and Scrum is neutral on requirement techniques. As the Scrum Primer^[20] states,

Product Backlog items are articulated in any way that is clear and sustainable. Contrary to popular misunderstanding, the Product Backlog does not contain "user stories"; it simply contains items. Those items can be expressed as user stories, use cases, or any other requirements approach that the group finds useful. But whatever the approach, most items should focus on delivering value to customers.

- Use cases are mainly diagrams. Larman stresses that "use cases are not diagrams, they are text"^[21]
- Use cases are always complex and difficult to write, understand and learn.

Tools

Text editors and/or word processors with template support are often used to write use cases. For large and complex system requirements, dedicated use case tools are helpful.

- CaseComplete
- Enterprise Architect
- MagicDraw
- Rational Software's RequisitePro - one of the early, well-known use case and requirement management tools in the 1990s.
- Wiki software - good tools for teams to author and manage use cases collaboratively.

Most UML Tools support both the text writing and visual modeling of use cases.

See also

- Abuse case
- Business case
- Event partitioning
- Feature
- List of UML tools
- Misuse case
- Requirement
- Scenario
- Storyboard
- Test Case
- Unified Process
- Use Case Points
- User story

References

1. ^ Jacobson et al, 1992.
2. ^ "Alistair Cockburn, "Use cases, ten years later" "
(http://alistair.cockburn.us/index.php/Use_cases%2C_ten_years_later). Alistair.cockburn.us. 2002. Retrieved 2013-04-17.
3. ^ Jacobson, Ivar; Spence, Ian; Bittner, Kurt (December 2011). "Use Case 2.0: The Guide to Succeeding with Use Cases" (<http://www.ivarjacobson.com/download.ashx?id=1282>). Ivar Jacobson International. Retrieved 2014-05-05.
4. ^ "Business Analysis Conference Europe 2011 - 26-28 September 2011, London, UK"
(<http://www.irmuk.co.uk/BA2011/>). Irmuk.co.uk. Retrieved 2013-04-17.
5. ^ *a b c d e f g h* Fowler, 2004.
6. ^ Cockburn, 2001
7. ^ *a b* Cockburn, 2001. Page 120.
8. ^ Cockburn, 2001. Inside rear cover. Field "Use Case Title".
9. ^ Alexander and Beus-Dukic, 2009. Page 121
10. ^ Cockburn, 2001. Inside front cover. Icons "Design Scope".
11. ^ Suzanne Robertson. *Scenarios in Requirements Discovery*. Chapter 3 in Alexander and Maiden, 2004. Pages 39-59.
12. ^ Cockburn, 2001. Inside front cover. Icons "Goal Level".
13. ^ *a b c d* Cockburn, 2001. Page 53.
14. ^ Cockburn, 2001. Page 55.
15. ^ *a b* Alexander and Beus-Dukic, 2009. Page 39.
16. ^ Cockburn, Alistair (2008-01-09). "Why I still use use cases"
(<http://alistair.cockburn.us/Why+I+still+use+use+cases>). *alistair.cockburn.us*.

17. ^ Meyer, 2000. (page needed)
18. ^ Armour and Miller, 2000. (page needed)
19. ^ Denney, 2005. (page needed)
20. ^ Pete Deemer, Gabrielle Benefield, Craig Larman, Bas Vodde (2012-12-17). "The Scrum Primer: A Lightweight Guide to the Theory and Practice of Scrum (Version 2.0)" (http://www.infoq.com/minibooks/Scrum_Primer). InfoQ.
21. ^ Larman, Craig. *Applying UML and patterns*. Prentice Hall. pp. 63 – 64. ISBN 0-13-148906-2.

Further reading

- Alexander, Ian, and Beus-Dukic, Ljerka. *Discovering Requirements: How to Specify Products and Services*. Wiley, 2009.
- Alexander, Ian, and Maiden, Neil. *Scenarios, Stories, Use Cases*. Wiley 2004.
- Armour, Frank, and Granville Miller. *Advanced Use Case Modeling: Software Systems*. Addison-Wesley, 2000.
- Kurt Bittner, Ian Spence, *Use Case Modeling*, Addison-Wesley Professional, Aug 20, 2002.
- Cockburn, Alistair. *Writing Effective Use Cases*. Addison-Wesley, 2001.
- Larry Constantine, Lucy Lockwood, *Software for Use: A Practical Guide to the Essential Models and Methods of Usage-Centered Design*, Addison-Wesley, 1999.
- Denney, Richard. *Succeeding with Use Cases: Working Smart to Deliver Quality*. Addison-Wesley, 2005.
- Fowler, Martin. *UML Distilled (Third Edition)*. Addison-Wesley, 2004.
- Jacobson Ivar, Christerson M., Jonsson P., Övergaard G., *Object-Oriented Software Engineering - A Use Case Driven Approach*, Addison-Wesley, 1992.
- Jacobson Ivar, Spence I., Bittner K. *Use Case 2.0: The Guide to Succeeding with Use Cases*, IJI SA, 2011.
- Dean Leffingwell, Don Widrig, *Managing Software Requirements: A Use Case Approach*, Addison-Wesley Professional. Dec 7, 2012.
- Meyer, Bertrand. *Object Oriented Software Construction. (2nd edition)*. Prentice Hall, 2000.
- Gunnar Overgaard, Karin Palmkvist, *Use Cases: Patterns and Blueprints*, Addison-Wesley Professional, Nov 12, 2004.
- Schneider, Geri and Winters, Jason P. *Applying Use Cases 2nd Edition: A Practical Guide*. Addison-Wesley, 2001.

External links

- Use Cases (Usability.gov) (http://www.usability.gov/methods/design_site/usecasesresource.html)

- Basic Use Case Template (http://alistair.cockburn.us/index.php/Basic_use_case_template) by Alistair Cockburn
- Application of use cases for stakeholder analysis "Project Icarus: Stakeholder Scenarios for an Interstellar Exploration Program", JBIS, 64, 224-233 (http://www.academia.edu/1354848/PROJECT_ICARUS_STAKEHOLDER_SCENARIOS_FOR_A_N_INTERSTELLAR_EXPLORATION_PROGRAM)
- "An Academic Survey on the Role of Use Cases in the UML" (<https://www.uleth.ca/survey/uml/>)

Retrieved from "http://en.wikipedia.org/w/index.php?title=Use_case&oldid=631461557"

Categories: [Software project management](#) | [Software requirements](#) | [Unified Modeling Language](#) | [Systems Modeling Language](#) | [1986 establishments in Sweden](#) | [1986 in computer science](#) | [Swedish inventions](#) | [Agile software development](#)

- This page was last modified on 28 October 2014 at 13:40.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.