# In Praise of Use Cases

Ian O'Neill
School of Electronics, Electrical Engineering and Computer Science
The Queen's University of Belfast
Belfast, BT7 1NN, N. Ireland
i.oneill@qub.ac.uk

***Abstract*— In this paper I argue that student software developers should be concerned not just with tersely expressed requirements ('user stories'), but with more involved 'use cases'. By encouraging developers to consider system behavior in some detail, the use-case approach lays the foundations for a coherent analysis of the object-based components that realize those behaviors. Though not objects in themselves, use cases encourage an object-oriented mindset. I consider some of the pitfalls that students must avoid when they are working with use cases, and propose an automated tutorial to help tackle some recurring difficulties.**

***Keywords— use cases, UML, user stories, agile, educational software***

## I. INTRODUCTION

In this paper, I examine some common problems associated with use cases in the classroom, while reminding educators of the advantages of including use cases in the software engineering (SE) process. I draw on my work teaching SE to groups of 200+ second-year students on undergraduate computing courses that last three to five years. For seasoned educators, some of my comments may simply confirm their own experiences. Others will find reminders of difficulties to avoid, and may be encouraged to adopt on-line tutorials, like the one outlined here, as a means of reinforcing key points. Sometimes the challenge of use cases is in the UML notation. Sometimes it involves the very meaning of the term 'use case'. I begin with the latter.

## II. WHY USE CASES (STILL)? – AND HOW THEY DIFFER FROM USER STORIES

Booch, Rumbaugh and Jacobson offer a definition that every student developer on a use case driven project should learn by heart: "A *use case* is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor" [1]. A common problem among student developers is the tendency to reduce the use case to a single button click or even an on-screen feature: the resulting use case diagram can resemble a decision diagram, where each use case is a step in a sequence, rather than a sequence in its own right. One likely source of confusion is the similarity (not least in the name!) between *use cases* on the one hand and, on the other, the *user stories* favored by agile methodologies, 'Agile' has gained a prominence that it did not have when Jacobson's influential text on use case driven software engineering [2] first appeared 25 years ago, so that now undergraduate students may learn about user stories before use cases. Tersely expressed user stories conceal different degrees of complexity. At one end of the scale are *epics* (e.g. "As a user, I can back up my entire hard drive[, so that...]" [3]) that must be decomposed into smaller, more workable user stories before coding begins. At the other extreme, the user story can indeed represent a 'single-click' operation ("As an estimator, I want to see the item we're estimating, so that I know what I'm giving an estimate for." [4]).

In his ACM webinar *Agile Methods: The Good, the Hype and the Ugly* [5], drawing on his book that covers many of the same themes [6], Bertrand Meyer succinctly identifies some fundamental problems associated with face-value implementation of user stories: their lack of abstraction draws the developer into providing a solution to the story 'as is', without considering the many variants that customers – other perhaps than the story's originator – are likely to require in their own particular circumstances, whether now or in the future; user stories may also, as separate elements in a product backlog, lull the developer into seeing every project as comprising only 'additive complexity' that can be built up largely in discrete layers (Meyer's 'lasagne' metaphor), as opposed to the all-too-frequent 'multiplicative complexity' (his 'linguine' metaphor), where an attempt to change a single piece of behavior is likely to affect many existing behaviors, like tugging on one strand in a tangle of pasta!

While they are not a cure-all for the problem of requirements representation, an important advantage of *use cases*, as part of an object-oriented (OO) development process, is that they split a development problem into substantial, well-considered 'chunks' of behavior, entailing an analysis of both main and alternative flows of actor-system interactions. The use case approach encourages the developer to work in an 'object-friendly' way: analyzing the written description of the flows in a use case helps the developer identify suites of collaborating objects that realize the behavior the customer needs, under typical conditions and when exceptions occur.

Moreover, use cases bring into focus the dependencies between different sets of system behaviors, dependencies that are likely to color the eventual system architecture. The written descriptions of the use cases – those sets of sequences of actions, with their references to other extending or included sets – are a more substantial record of required behavior than the simple diagrammatic notation that shows use cases in context. But even that simple notation, with its connected use case ellipses and stick-figure actors, has value in assisting interested parties understand, in broad brush strokes, how one set of behaviors relates to, and potentially impacts on, other sets of behaviors; it also shows at a glance who (or what) might be expected to make use of those behaviors. The written descriptions put detail on the individual behaviors to be developed; the very accessible graphical notation stimulates and aids discussion of how the system should behave as a whole.

As incorporated into the UML [7], the use case based approach to software development already had much to commend it. However, reacting to more recent developments in the software industry, Jacobson has also proposed an accommodation between use cases and agile's user stories: in his *Use-Case 2.0* approach [8] [9], use cases are divided into *use case slices* that correspond to one or more *stories* (similar to the *user stories* of Scrum, etc.).

*Use-Case 2.0* makes a well-argued case for the complementarity of use cases and the stories that form the basis of the dialogue between developers and stakeholders. Likewise, student developers just have to understand, and use to their advantage, the differences between the techniques of agile and the conventions of the UML.

## III. GETTING USE CASE DIAGRAMS RIGHT

The UML offers the student some flexibility in the way he or she describes a flow of events within a use case [10] (e.g. bulleted descriptions of normal and alternative flows, pre- and post-conditions, identification of extension points), but it specifically defines the diagrammatic notation for representing relationships between the use cases. The intention is that colleagues and clients, or students and teachers, should be able to interpret the diagram quickly but accurately. Accuracy of interpretation requires that the three relationships be used carefully and consistently (definitions of <<extend>> and <<include>> especially have been a recurring subject of debate – e.g. [11]):

*1)* Generalization, represented by an open-headed arrow with a solid-line tail. In this relationship the more general use case is at the arrowhead, while the more specialized use case is at the tail (M and J are specializations of H in Figure 1: where H is allowed, M or J can be used). Using the correct arrow, and positioning the correct use cases at the head and tail, are important details for the student to get right.

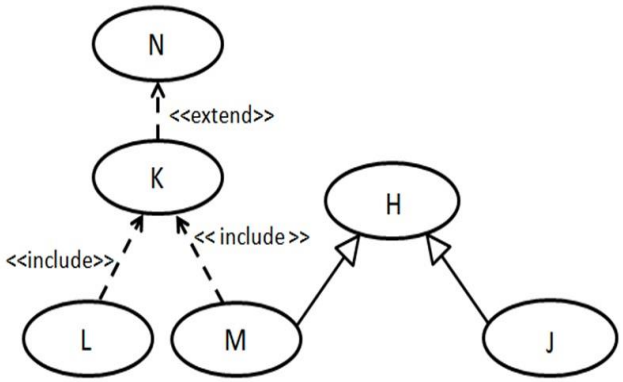


Fig. 1. A simple UML use case diagram.

*2)* <<include>>. According to Booch, Rumbaugh and Jacobson [12], "an include relationship between use cases means that the base use case explicitly incorporates the behavior of another use case at a location specified in the base. The included use case never stands alone, but is only instantiated as part of some larger base that includes it." This still leaves a degree of ambiguity, for it is not made explicit whether the 'location specified in the base' must be in the main flow of the base use case or if it may be in an alternative flow. Often trainers give their students less room for maneuver in the use of <<include>>. For example, in its blog, Requirements Inc. [13] maintains "A includes B […] – A cannot produce success outcome without running B". See also Dennis, Wixom and Tegarden's [14] association of <<include>> with a use case's normal flow. In explaining the <<include>> relationship to my students, I have favored the more prescriptive approach, so that from the include relationship between L and K shown in Figure 1, it is understood that Use Case L, if it executes normally, always includes Use Case K in its main (or normal) flow. This makes it easier to distinguish <<include>> from <<extend>> which follows.

*3)* Unlike <<include>>, then, <<extend>> represents optional or conditional behavior: "The base use case may stand alone, but under certain conditions its behavior may be extended by the behavior of another use case" [15]. (Some students will struggle initially to appreciate that <<extend>> differs from `extends`, the reserved word used to signal *inheritance* in OO languages like Java.) Like <<include>>, the <<extend>> relationship needs to be read in the direction of the dashed stick-arrow, where the use case that provides the 'extending' or conditional behavior is at the tail and the use case that is 'being extended' is at the head of the arrow. Thus, in Figure 1, the K-N relationship reads: 'Use Case K extends Use Case N'. Very often in their diagrams, students place the <<extend>> arrow the wrong way round, confused by the fact that if 'K extends N', the functionality of K will be accessed from an 'extension point' in a flow of events in N (i.e., put simply, N 'invokes' K).

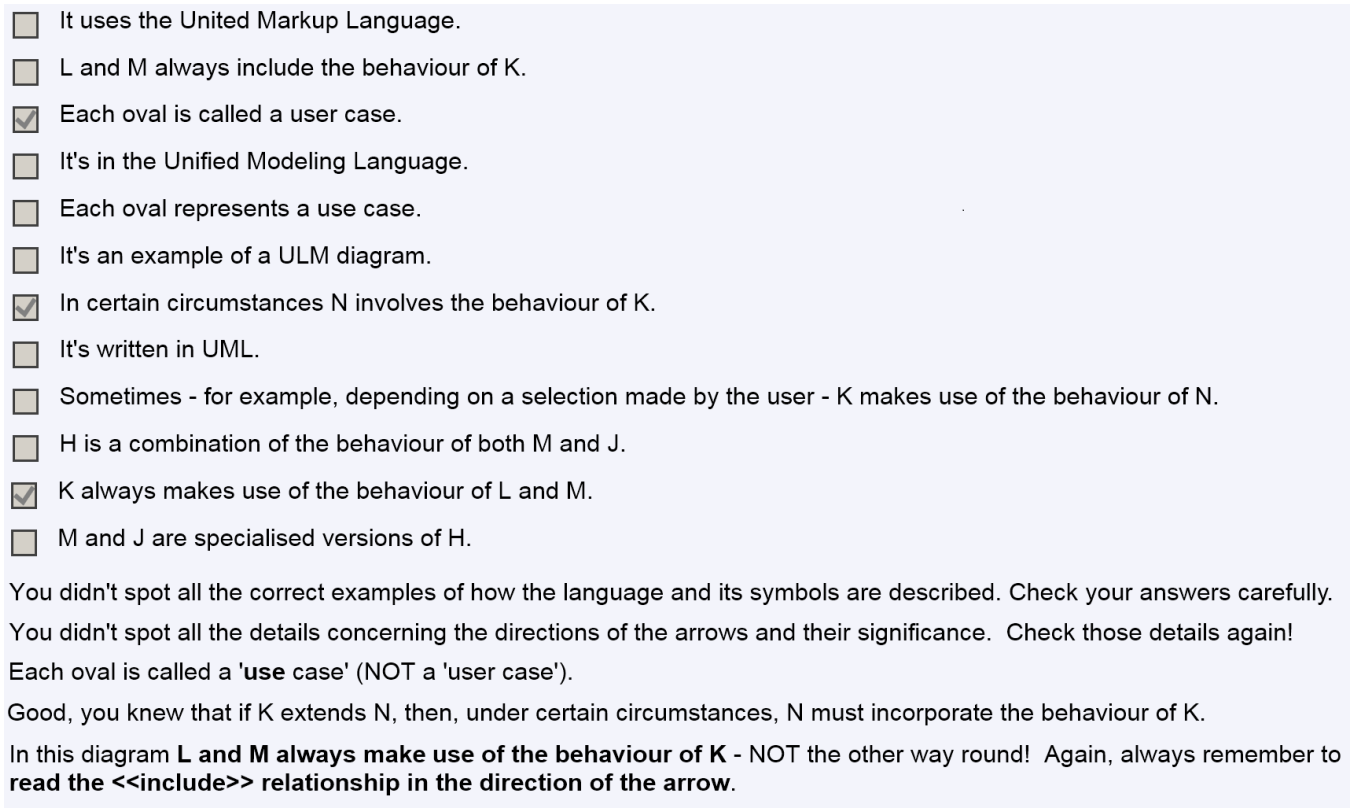


Fig. 2. What can you tell me about Figure 1? The student responds and system corrects.

## IV. Educational software (and some Related Work) to the rescue

For the last two years, in the lead-in to their end-of-year exam, I have provided students with a short, adaptive, on-line 'use case tutorial'. Both exam and tutorial are written in Questionmark, a widely-used commercial software package for implementing and scheduling multiple-choice/multiple-response assessments [16]. Other multiple-choice/multiple-response software is available, sometimes as freeware (e.g. [17]); EduSymp attendees will be familiar with e-learning issues more generally (e.g. [18]). However, this use case tutorial specifically exploits the often neglected, but powerful, conditional 'jump blocks' in Questionmark, which can be used to match the system's follow-up questions to a specific combination of student responses. Drawing on my research into object-based natural language dialogue systems [19], I have used these jump blocks extensively to give the student-system interaction a novel, 'conversational' twist. The interaction resembles a branching, frame-based natural language dialogue [20], where a 'frame' or 'set' of responses determines the next step in the conversation. In a separate article (to appear in *ACM Inroads*) I detail the dialogue management strategy. In this instance, the student comments on a use case diagram (the example in Figure 1) and the system responds. Figure 2 shows the system's reaction to an incomplete and inaccurate set of student responses – given this response combination, the system will give feedback and then simply prompt the student to try again. If the student's response combinations indicate that he or she is having difficulty in a particular area (terminology, or interpretation of the symbols), the system will quiz the student on the problem area alone, providing feedback on each new set of responses, and moving on only when the responses are sufficiently accurate. The tutorial concludes when the student gives a good, broad-ranging set of responses. Tenacious but helpful in its comments, the system evokes the small-scale, person-to-person tutorials of earlier years.

Student feedback on the automated tutorial has been very encouraging. Comments taken from the surveys (also implemented in Questionmark) that accompany the tutorial include: "I thought that the way that you could repeat the questions again if you didn't pass was good, and I thought that the personalized feedback was great and really helpful"; "It was good, told me what I didn't understand and confirmed what I already knew"; "I thought it explained it quite well, and wasn't too critical on my wrong answers, which I liked a lot". The main requests from the surveys are for longer tutorials, and tutorials on other subjects. Having trialed the short use case tutorial as a revision aid, I hope to introduce it earlier in the academic year, where it will be a ready reminder of some key points as students embark on a substantial use case driven SE project.

## V. Concluding remarks

On traditional OOSE pathways, when students write their 'final-year dissertation', they are expected to describe their

software solution and justify their approach. A use case model gives clear structure to the required behavior and so to the development process – provided the student, with the educator's help, has learnt to avoid common conceptual and notational pitfalls associated with use cases. Automated tutorials can support that learning process. In software engineering, use cases encourage developers and clients to think beyond isolated actions and consider the system in the round, from an early stage of the development process.

Use cases may have a similar effect in life more generally! At a talk I attended in London in the 1990s (sadly I do not have a more detailed reference), Jacobson told how use cases had even influenced the layout of his kitchen. So, if *Make toast* is a use case, realize it properly: make sure that your bread bin, toaster, cutlery drawer and fridge with the butter in it are all beside each other!

## REFERENCES


[1] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, 2nd ed. Upper Saddle River: Addison-Wesley, 2005, p. 228.

[2] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, Object-Oriented Software Engineering – A Use Case Driven Approach. Wokingham: ACM Press/Addison-Wesley, 1992.

[3] Mountain Goat Software, https://www.mountaingoatsoftware.com/agile/user-stories#section-3, last accessed 2017/06/27.

[4] Mountain Goat Software, https://www.mountaingoatsoftware.com/blog/advantages-of-the-as-a-user-i-want-user-story-template, last accessed 2017/06/27.

[5] B. Meyer, Agile Methods: The Good, the Hype and the Ugly, ACM Webinar, 18 February 2015, https://www.youtube.com/watch?v=ffkIQrq-m34, last visited 2017/10/06

[6] B. Meyer, Agile! The Good, the Hype and the Ugly. Cham, Switzerland: Springer International Publishing, 2014.

[7] G. Booch, J. Rumbaugh, and I. Jacobson, *ibid.*, Chapters 17 and 18.

[8] I. Jacobson, I. Spence, and K. Bittner, Use-Case 2.0 – The Guide to Succeeding with Use Cases, https://www.ivarjacobson.com/publications/white-papers/use-case-ebook, last accessed 2017/08/07 .

[9] I. Jacobson, I. Spence, and B. Kerr, "Use Case 2.0 – the hub of software development," ACM Queue, 4 (1), 2016, pp. 94-122.

[10] G. Booch, J. Rumbaugh, and I. Jacobson, *ibid.*, p. 230.

[11] M.A. Laguna and J.M. Marqués, "On the multiplicity semantics of the extend relationship in use case models," in Software and Data Technologies, Third International Conference, ICSOFT 2008, J. Cordeiro, B. Shishkov, A. Ranchordas, and M. Helfert, Eds. Heidelberg: Springer, 2009, pp. 62–75.

[12] G. Booch, J. Rumbaugh, and I. Jacobson, *ibid.*, p.233.

[13] Requirements Inc Blog, http://requirementsinc.com/uml-basics-include-and-extend-stereotypes-in-use-cases/, last accessed 2017/06/27.

[14] A. Dennis, B.H. Wixom, and D. Tegarden, Systems Analysis & Design – An Object-Oriented Approach with UML, 5th ed. Hoboken, N.J.: Wiley, 2015.

[15] G. Booch, J. Rumbaugh, and I. Jacobson, *ibid.*, p. 234.

[16] Questionmark Homepage, https://www.questionmark.com/, last accessed 2017/06/27.

[17] Moodle, https://www.moodle.org, last accessed 2017/08/08.

[18] D.R. Stikkolorum, B. Demuth, V. Zaytsev, F. Boulanger, and J. Gray, "The MOOC hype: can we ignore it? Reflections on the current use of massive open online courses in Software Modeling Education," in MODELS Educators Symposium, EduSymp 2014, B. Demuth and D.R. Stikkolorum, Eds. Valencia, Spain, September 2014, pp. 75-86, http://ceur-ws.org/Vol-1346, last accessed 2017/08/07.

[19] I. O'Neill, P. Hanna, X. Liu, D. Greer, and M.F. McTear, "Implementing advanced spoken dialogue management in Java," Science of Computer Programming, 54 (1), pp. 99-124, 2005.

[20] M.F. McTear, Spoken Dialogue Technology: Toward the Conversational User Interface. London: Springer-Verlag, 2004.