**QUEEN'S
UNIVERSITY
BELFAST**

In praise of use cases – a paean with a software accompaniment

O'Neill, I. (2018). In praise of use cases – a paean with a software accompaniment. *Computer Science Education*, 1-23. <https://doi.org/10.1080/08993408.2018.1472949>

Published in:
Computer Science Education

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
[Link to publication record in Queen's University Belfast Research Portal](#)

Publisher rights

© 2018 Informa UK Limited, trading as Taylor & Francis Group. This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights

Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

In Praise of Use Cases – a Paean with a Software Accompaniment

This article reminds readers of the benefits that use cases bring to the software development process. Use cases, as featured in the UML (Unified Modeling Language), are contrasted with the much terser “user stories” favoured by agile methods. With their normal and alternative flows, and extending and included behaviours, use cases encourage developers to consider actor–system interaction in detail, preparing the way for coherent mechanisms of interacting and inheriting objects that realise the required functionality. The textual and visual representation of use cases has a simplicity that encourages discussion of requirements among developers, and between developers and clients, but only if “use case basics” are understood and applied consistently. An innovative use of educational software is proposed, to alleviate some fundamental but recurring difficulties, and to give students in large cohorts the benefit of focussed tuition and feedback. The approach will appeal to educators in software engineering and beyond.

Keywords: use cases; unified modelling language (UML); user stories; agile; educational software

1. Introduction

This article examines some common problems associated with use cases in the classroom, while reminding educators of the advantages that use cases bring to the software engineering (SE) process. Use cases are a proven approach to considering software requirements in the round: what are the outcomes of value that a system should produce; what are the main and conditional behaviours of the system that achieve those outcomes; and how do the system’s different behaviours relate to each other (Podeswa, 2010; Wiegers & Beatty, 2013; Dennis, Wixom & Tegarden, 2015)? Answering these questions correctly sets the tone for the software that follows, in terms of both its functionality and its architecture. Eliciting and analysing system requirements remains a fundamental challenge in the software development process: if it is done incorrectly,

the resulting software either does not work, or works only in a very limited set of circumstances with little scope for future development.

The present article draws on the author's experience of teaching SE to groups of 200+ second-year students on undergraduate computing courses that last three to five years. For seasoned educators, some of the comments expressed here will simply confirm their own experiences. Others will find reminders of difficulties to avoid, and may be encouraged to adopt automated on-line tutorials, like the one outlined here, as a means of reinforcing key points. This automated tutorial is not intended to solve all the problems that the educator encounters when teaching use case fundamentals, but it does present an approach to the problem that, in the light of favourable student reaction, holds much promise (and naturally is applicable to areas other than software engineering). Based on *In Praise of Use Cases*, presented at *EduSymp 2017* (O'Neill, 2017), this more substantial article gives further details of the automated tutorial and the dialogue-based strategy that drives it; it gives more coverage of the GUI-based interaction between student and system and presents more of the students' feedback. Furthermore, this extended article gives additional examples of researchers' attempts to reconcile *use cases* and *user stories*. Finally it considers how one cohort of students, once they have grasped the basics, go on to apply their knowledge of use cases to their project work.

With use cases, sometimes the challenge is how to represent them in the UML notation, which, simple as it is, must be used consistently and in the generally accepted manner if it is to serve as a *lingua franca* among developers and between developers and clients. Sometimes, however, the problem involves the very term 'use case' – whether it is a matter of writing the term accurately (*use cases*, rather than 'user cases', as this educator's students often say initially), or whether it is simply – and more

importantly – a matter of understanding what a use case is in the context of the UML, and what it is not. Let us begin by considering the definition of a use case.

2. How use cases differ from user stories

2.1. Describing use cases and 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.” (Booch, Rumbaugh, & Jacobson, 2005, p. 228)

A common problem among student developers is the tendency to reduce the use case to a single button click (e.g. use case ‘Navigate to home page’) or even an on-screen feature (e.g. <<include>> use case ‘Tutor list’): the resulting use case diagram can resemble a cross between an ER diagram and a decision diagram, to the extent that extraneous ‘decision diamonds’ appear amidst the use case ellipses and their connecting arrows. One likely factor contributing to the problem of granularity is the similarity (not least in the name!) between *use cases* on the one hand and, on the other, the *user stories* favoured by agile methodologies – thus, perhaps, the frequent occurrence of the erroneous term ‘user case’. Agile has gained a prominence that it did not have when Jacobson’s influential text on use case driven software engineering (Jacobson, Christerson, Jonsson, & Övergaard, 1992) first appeared a quarter of a century ago, so that now undergraduate students regularly 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...]”

(Mountain Goat Software [MGS], 2017) 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.” (Cohn, 2017)).

In his February 2015 ACM webinar *Agile Methods: The Good, the Hype and the Ugly* (Meyer, 2015), drawing on his book that covers many of the same themes (Meyer, 2014), 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 (as Meyer puts it) ‘additive complexity’ that can be built up in discrete layers (his ‘lasagne’ metaphor), as opposed to the all-too-frequent ‘multiplicative complexity’ (his ‘linguine’ metaphor), where an attempt to change a single piece of behaviour affects many existing behaviours, like tugging on one strand in a tangle of pasta!

2.2. Some use case advantages

2.2.1. Watching out for optional and exceptional behaviour

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 problem into substantial, well-considered ‘chunks’ of behaviour, unlike the much terser user stories. Meyer too recognises this distinction –

“A user story is the description of a fine-grain functionality of the system, as seen by its users. The more general notion is use case [...]. A use case can be big: it describes an entire interaction scenario, for example the process of ordering an item on an e-commerce site. A user story is much smaller.” (Meyer, 2014, p.119)

– though elsewhere he tars use cases and user stories with the same brush:

“When using software systems [...] have you ever felt like Tintin the day he was being marched in a straightjacket? As soon as you dare to depart from the exact scenario that the designers [...] have planned for you, nothing works any more. This kind of system is the direct result of requirements based on the sole analysis of use cases or user stories” (Meyer, 2014, p. 13)¹

Of course, if taken too simplistically – i.e. “to achieve *this*, the user need only do *that*” – both *user stories* and *use cases* will lead to systems of very limited and limiting functionality. However, the point about *use cases* is that *each* use case represents “a *set* of sequences of actions” (that yields an *observable result of benefit* to an actor): in other words, once the use case is finished, the actor – usually a person, and usually fulfilling a real-world role – will move on, knowing that he or she has achieved something useful (“I’m now registered with the hotel booking system”; “I’ve now changed my account details”; etc.). To be able to deliver that *set* of sequences of actions, the software engineer (acting as analyst) needs to examine the basic user requirements closely. At any of the steps in a flow of events, could something go wrong or should there be optional system behaviour? If, so, is there an appropriate alternative flow, or another *extending* use case? Some of these embellishments of the ‘happy path’, or deviations

¹ Though rather more sympathetic towards both use cases and user stories than Meyer, the team of Oran, Nascimento, Santos and Conte (2012) also conclude that there is little difference in the effectiveness of the two approaches in communicating requirements.

from it, will be a matter of common sense, but others may emerge only as the developer submits his or her analyses to closer scrutiny, by the customer or by the eventual end-user of the system (e.g. “What happens if I want to save a copy of my details as a PDF?”). Meeting the business-related requirements of customers and the job-related requirements of end-users – in *all* reasonable circumstances – is a fundamental objective of software engineering, and use cases encourage such an approach.

2.2.2. A good basis for OO development

From a methodological perspective, the use case approach, though not in itself dealing with classes or objects, also encourages the developer to work in an ‘object-friendly’ way. The written descriptions of the use cases – with their references to other extending, included or generalised use cases – are the detailed record of the required behaviour. Analysis of the descriptions helps the developer identify the suites of inheriting and collaborating software objects that will realise the behaviour, under typical conditions and when exceptions occur.

2.2.3 An accessible diagrammatic notation

Complementing those descriptions is the very simple diagrammatic UML notation – with its connected use case ellipses and stick-figure actors – that shows the use cases in context. Simple as it is, that notation has value in assisting interested parties understand at a glance how one set of behaviours relates to, and potentially impacts on, other sets of behaviours; likewise it shows who (or what) might be expected to make use of those behaviours. The written descriptions put detail on the individual behaviours to be developed; the very accessible graphical UML notation stimulates and aids discussion of how the system should behave as a whole. This very immediate, diagrammatic overview of intended system behaviour is especially useful in the case of university

work, where the academic supervisor or assessor may have responsibility for many different student projects in a year group and must understand the aims, objectives and overall structure of all of them.

3. Supporting use cases with user stories

Various efforts have been and are being made to reconcile use cases and (user) stories.

3.1 Jacobson's Use-Case 2.0

In Jacobson's *Use-Case 2.0* approach (Jacobson, Spence, & Bittner, 2011; Jacobson, Spence, & Kerr, 2016) use cases are divided into *use case slices*, suitably sized work items that correspond to one or more *stories* (similar to the *user stories* of Scrum, etc., but with a very specific goal – e.g. “‘Withdraw a standard amount of \$100,’ ‘Withdraw a non-standard amount of \$75 and get a receipt,’ (Jacobson et al. (2016, p. 106)). A particular development increment may involve work on slices taken from several use cases, each slice supporting one or more stories. *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.

3.2 Other bridge-building attempts

A survey of the literature reveals much activity in this area. For example, Firesmith (2004) already identifies simple abstract *stories* along with detailed, case-specific *scenarios* as the raw materials from which *use case paths* or ‘flows’, user-system *use case path interactions*, and ultimately trigger- and precondition-rich *textual requirements* are derived. Elsewhere the approach to melding stories and use cases demonstrates an almost instinctive simplicity: Śmiałek, Jarzębowski and Nowakowski (2012) formulate stories in the form *subject-verb-(direct)object* or *subject-verb-object-(indirect)object*, and use these directly as individual steps that collectively make up a

flow – whether normal or conditional – in a particular use case. Wautelet, Heng, Hintea, Kolp and Poelmans (2016) are more forthright in their ambition to ‘bridge user story sets with the use case model’, identifying ‘goals’ and ‘tasks’ in user stories and mapping these to use cases that are linked as appropriate with <<extend>> and <<include>> relationships (the authors provide a detailed rationale to justify the mapping process and the use of the standard relationships).

4. Getting use case diagrams right

Identifying and describing use cases is one matter; identifying and describing them as unambiguously as possible is another. The written description of each use case is of course paramount, and the UML offers the student some flexibility in the way he or she describes the flows of events within a use case (Booch et al., 2005) (e.g. bulleted descriptions of normal and alternative flows, pre- and post-conditions, identification of extension points, identification of points at which other use cases are to be included in a flow) – see Section 7.2 also. However, the UML specifically defines the diagrammatic notation, simple as it is, for representing relationships between the use cases. For a use case diagram (like the one shown in Figure 1) to be interpreted quickly but accurately, the three relationships that the use case diagram can represent must be drawn carefully

Figure 1. A simple use case diagram.

and consistently²:

- (I.) 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.
- (II.) <<include>>. According to Booch, Rumbaugh and Jacobson (2005, p. 230), “an include relationship between use cases means that the base use case explicitly incorporates the behaviour 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 manoeuvre in the use of <<include>>. For example, in its blog, Requirements Inc. maintains “A includes B [...] – A cannot produce success outcome without running B” (<http://requirementsinc.com/uml-basics-include-and-extend-stereotypes-in-use-cases/>). Similarly, Dennis, Wixom and Tegarden (2015) associate <<include>> with a use case’s normal flow. In explaining the <<include>> relationship, the present author favours this

² Definitions of <<extend>> and <<include>> especially have been a recurring subject of debate – e.g. Laguna and Marqués (2009).

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.

(III.) Unlike <<include>>, then, <<extend>> represents optional or conditional behaviour: “The base use case may stand alone, but under certain conditions its behavior may be extended by the behavior of another use case” (Booch et al., 2005, p. 234). (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 behaviour 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).

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

5.1. Questionmark

And so to the original software that accompanies this review of the merits of use cases. How can knowledge of the basic relationships and other UML fundamentals be reinforced in a way that is convenient to student and lecturer alike? For the last two years, the author has provided second-year students with a short, adaptive, on-line ‘use

case tutorial’ written in Questionmark, a widely-used commercial software package for implementing and scheduling multiple-choice/multiple-response quizzes, surveys and exams (<https://www.questionmark.com/>). Questionmark will be familiar to many educators as the institutional technology of choice for assessment of large cohorts. To make exam announcements and scheduling more convenient, it is generally closely integrated with electronic student registers and module listings. Other multiple-choice/multiple-response software is available, sometimes as freeware – the case with another widely used learning platform, Moodle (<https://moodle.org>).

5.2. Setting the use of educational software in context

The role of software in teaching modelling has been discussed previously at EduSymp symposia (e.g. Stikkolorum, Demuth, Zaytsev, Boulanger, & Gray, 2014) and the discussion continues in this Special Issue. Educational software has a much longer history, part of a technological development that has led from the mechanical multiple-choice devices of the 1920s (Pressy, 1926), through use of electronic digital computers for Computer-Assisted Instruction in the 1960s (Suppes, Jerman, & Brian, 1968), to the ‘intelligent’, ‘adaptive’ and ‘conversational’ tutoring systems of the 21st century (some examples of which are considered by Rus, D’Mello, Hu and Graesser (2013)). Suffice to say that the automated, adaptive use case tutorial described here specifically exploits the often neglected, but powerful, conditional ‘jump blocks’ of the now well-established and functionally rich Questionmark system³: these jump blocks can be used to match a system’s follow-up questions to a specific combination of student responses, giving a novel conversational twist to the students’ interaction with the multiple-response

³ Questionmark’s large feature set, when compared with rival products, is confirmed by a recent survey of such systems (Fuentes, García, Ramírez-Gómez, & Ayuga, 2014).

What can you tell me about this diagram, which in Software Engineering is used to represent user requirements? Choose as many answers as you believe to be appropriate.

- ☐ It uses the United Markup Language.
- ☒ L and M always include the behaviour of K.
- ☐ Each oval is called a user case.
- ☒ It's in the Unified Modeling Language.
- ☒ Each oval represents a use case.
- ☐ It's an example of a ULM diagram.
- ☒ In certain circumstances N involves the behaviour of K.
- ☒ It's written in UML.
- ☐ Sometimes - for example, depending on a selection made by the user - K makes use of the behaviour of N.
- ☐ H is a combination of the behaviour of both M and J.
- ☐ K always makes use of the behaviour of L and M.
- ☒ M and J are specialised versions of H.

Well done. You know all the basics. You may wish to look at Booch, Rumbaugh and Jacobson's *The Unified Modeling Language User Guide* (Addison-Wesley) for more detailed information on the UML.

Figure 2. All the correct responses selected and the system's acknowledgement.

educational software. The interaction is in fact a *frame-based natural language dialogue*, where a 'frame' or 'set' of responses determines the next step in the conversation. O'Neill, Hanna, Liu, Greer and McTear (2005) and O'Neill and McTear (2000) describe examples of frame-based dialogue systems – more on the frame-based approach presently.

5.3. A dialogue strategy

In the automated tutorial associated with this paper, the student, by selecting check boxes, comments on a use case diagram (the example in Figure 1) and the system makes its next moves(s) on the basis of those comments. Figure 2 shows the system's stimulus, the student's responses – all the correct answers were selected – and the system's feedback (in the manner of point (2) (d) below). The underlying dialogue strategy is presented in more abstract, topic-independent form in a separate article in *ACM Inroads* (O'Neill, 2018)). Where use cases and the UML are concerned, the strategy can be summarised as follows:

- (1) The student is invited to comment on a simple use case diagram and is offered a selection of possible responses, some of which are correct, some incorrect. The responses, in this instance, cover two areas of interest that are associated with use cases: use of UML-related terminology; and interpretation of the UML notation⁴. The student makes his or her response selection, choosing as many or as few of the candidate responses as he/she thinks applicable to the diagram displayed.
- (2) The system makes its next dialogue move:
- (a) If the student's responses are generally poor (indicated by a behind-the-scenes overall score, calculated from the number of correct and incorrect answers chosen by the student), then the student is asked to try the exercise again.
 - (b) If the student's responses are generally good (but not perfect), the system congratulates the student, concludes the exercise, but indicates that he/she could do better and might like to try the exercise again to improve their performance.
 - (c) If the student fails completely in a particular area (in this case one of the two areas under consideration: *terminology* or *notation*), by choosing all of the incorrect responses and none of the correct responses in that area, then the system focuses on that area, presenting a reduced set of response

⁴ To simplify the dialogue interaction, only two 'areas of interest' were selected for this concept demonstrator. However, these areas illustrate the principle that certain answer combinations can be associated with particular 'problem areas', which an experienced tutor will recognise, and for which follow-up dialogue may be required.

options relevant to that area alone. Only when the student has performed sufficiently well in the area in isolation does the system offer the full set of options again.

- (d) If the student chooses all the correct options from the full set, the system congratulates the student warmly and suggests further reading that will widen his or her mastery of the UML.

As part of its response to the student's selections, and before it moves on to any new challenge, the system provides comprehensive feedback, commenting on the student's overall performance in each of the topic areas, confirming, with a reinforcing paraphrase, each of the student's correct responses, and correcting, with words of explanation, each of the student's incorrect responses. It is this combination of responses and feedback that gives the exercise its conversational flavour. Tenacious but supportive, the system evokes the small-scale, person-to-person tutorials of earlier years, now an all but unattainable luxury in institutions of higher education where an individual lecturer may be responsible for many very large classes.

5.4. When the answers are not quite right...

Let us briefly examine a more problematic example of the automated student-system dialogue – the case where the student falls well short of the ideal of a perfect set of responses! Figure 3 shows the system's reaction to just such an incomplete and inaccurate set of responses: given this response combination, the system will give feedback and then simply prompt the student to try again – its implementation of strategy (2) (a) above.

☒ It uses the United Markup Language.
☐ L and M always include the behaviour of K.
☐ Each oval is called a user case.
☐ It's in the Unified Modeling Language.
☐ Each oval represents a use case.
☒ It's an example of a ULM diagram.
☐ In certain circumstances N involves the behaviour of K.
☐ It's written in UML.
☐ Sometimes - for example, depending on a selection made by the user - K makes use of the behaviour of N.
☒ H is a combination of the behaviour of both M and J.
☐ K always makes use of the behaviour of L and M.
☐ M and J are specialised versions of H.

You didn't spot all the correct examples of how the language and its symbols are described. Check your answers carefully!
 You didn't spot all the details concerning the directions of the arrows and their significance. Check those details again!
 It's the '**Unified Modeling** Language' - NOT 'United Markup Language'!
 We talk about **UML** diagrams, from **U**nified **M**odeling **L**anguage (NOT ULM diagrams - whatever those might be!).
 In the diagram, **M** and **J** are **specialised examples of H** - they can be said to **inherit from H**. (The answer you selected implies the opposite - that H somehow acquires behaviour from the other two, which is incorrect.)

Figure 3. An incomplete and inaccurate set of responses from the student, and the system's response.

Or again, if the student fails completely in a topic area, notation say – the outcome illustrated by Figure 4 – the system, having given the student detailed feedback, will test the student once more in the area of weakness, presenting a much more limited range of candidate responses, specific to the area of difficulty (Figure 5). This movement towards a more tightly focused student-system dialogue is the system's implementation of strategy (2) (c).

- ☐ It uses the United Markup Language.
- ☐ L and M always include the behaviour of K.
- ☐ Each oval is called a user case.
- ☐ It's in the Unified Modeling Language.
- ☐ Each oval represents a use case.
- ☐ It's an example of a ULM diagram.
- ☐ In certain circumstances N involves the behaviour of K.
- ☐ It's written in UML.
- ☒ Sometimes - for example, depending on a selection made by the user - K makes use of the behaviour of N.
- ☒ H is a combination of the behaviour of both M and J.
- ☒ K always makes use of the behaviour of L and M.
- ☐ M and J are specialised versions of H.

You didn't spot all the correct examples of how the language and its symbols are described. Check your answers carefully!

You didn't spot all the details concerning the directions of the arrows and their significance. Check those details again!

The questions relating to the arrows caused problems.

Be very careful about the directions of the arrows and their significance.

Read the <<include>> and <<extend>> relationships in the direction of the arrow.

L <<include>> -> K, means that L includes the behaviour of K. K <<extend>> -> N, means that, under certain circumstances, K extends the behaviour of N (or, in other words, that under certain circumstances N involves the behaviour of K.)

The white-headed arrow indicates generalisation or inheritance, with the more general behaviour at the head of the arrow, and the inheriting (or more specialised) behaviour at the tail.

We'll try the arrow-related questions again in a moment.

In this diagram, under certain circumstances, **N makes use of K** - in other words, under certain circumstances '**K extends N**'. This is the **opposite** of saying that K makes use of N, as you did! Always remember to **read the <<extend>> relationship in the direction of the arrow**.

In the diagram, **M and J are specialised examples of H** - they can be said to **inherit from H**. (The answer you selected implies the opposite - that H somehow acquires behaviour from the other two, which is incorrect.)

In this diagram **L and M always makes use of the behaviour of K** - NOT the other way round! Again, always remember to **read the <<include>> relationship in the direction of the arrow**.

Figure 4. The student selects all the incorrect notation-related responses and none of the correct ones. – The system gives its feedback.

What can you tell me about this diagram, which in Software Engineering is used to represent user requirements? Let's concentrate on the arrows and the relationships they represent. Choose as many answers as you believe to be appropriate.

- ☐ L and M always include the behaviour of K.
- ☐ In certain circumstances N involves the behaviour of K.
- ☐ Sometimes - for example, depending on a selection made by the user - K makes use of the behaviour of N.
- ☐ H is a combination of the behaviour of both M and J.
- ☐ K always makes use of the behaviour of L and M.
- ☐ M and J are specialised versions of H.

Figure 5. A more focussed stimulus from the system – concentrating on the area of weakness.

5.5. A closer look at the frame-based approach

Notionally, for each of the student's dialogue turns, the system makes available several slots in a dialogue frame, each slot corresponding to one of the possible responses that the student might give. In this case, the student's responses will either fill a slot (the student has selected that response) or leave a slot empty (the student has not selected that response). Slots that are selected may be viewed as a logical TRUE, and slots that are not selected as a logical FALSE. Given that some of the slots represent responses that are either factually correct or incorrect (in this case the system is asking the student to comment on a simple use case diagram), it is then possible to use certain distinctive TRUE/FALSE combinations to determine the system's next move in a selection of key circumstances – if, for example, the student has selected all the incorrect notation-related answers (all the *incorrect* notation-related answer slots are *filled* – they are logically TRUE) and has omitted all the correct notation-related answers (all the *correct* notation-related answer slots are *empty* – they are logically FALSE), then the system will proceed by examining the area of notation alone. Distinctive combinations or patterns of responses will be familiar to the experienced instructor, and may, as in this instance, indicate a student who is struggling with basic concepts. Like the instructor, the system can take appropriate action, giving constructive and corrective feedback, before asking the student to try again, with the focus now on the area of concern.

Turn 1 The system invites the student to comment on a UML diagram and offers a selection of possible terminology- and notation-related responses. The student may select as many or as few responses as he/she wishes...

[**Note:** In the representation of the 'dialogue frame' below, X_T is a terminology-related response. X_N is a notation-related response. Correct responses are marked ✓; incorrect responses are marked ✗. Responses selected by the student are shown thus: i.e. a 'slot' in the dialogue frame has been filled!]

In this case, the student chooses all the incorrect notation-related responses and does not select any other responses.

Turn 2 Having pointed out that correct responses were missed, the system gives specific feedback on the incorrectly selected notation-related responses, and indicates that it will re-examine notation. It invites the student to comment on the UML diagram again, but this time offers only the notation-related responses...

Now the student chooses all the correct notation-related responses and none of the incorrect ones.

Turn 3 The system congratulates the student on his/her notation-related knowledge and asks the student to comment on the UML diagram once more: now it offers all the possible responses, both terminology- and notation-related...

The student chooses all the correct responses.

Turn 4 The system congratulates the student again and announces the end of the tutorial. It offers further reading recommendations.

Figure 6. The student-system interaction viewed as a frame-based dialogue.

Of course, if a rigorous ‘truth-table’ approach were to be adopted, the number of the system’s next possible moves would grow exponentially (2^n) as the number of possible response selections (n) increased. To avoid this difficulty, only the most distinctive combinations of student responses need be chosen for special treatment (a ‘simplified’ approach in that respect), while other generally ‘good’ or ‘poor’ performances can be assessed by totalling correctly selected and incorrectly selected responses, where a correctly selected response is worth +1 and an incorrectly selected response is worth -1.

Figure 6 illustrates the frame-based process, taking the example of a student who:-

- first misses many correct answers, but notably ‘fails completely on notation’, in *Turn 1*;
- then, having taken into account the feedback from the system, succeeds in *Turn 2*, by selecting all the correct ‘notation only’ responses;
- is given a chance to repeat the exercise as a whole in *Turn 3*, this time succeeding admirably; and
- is congratulated in *Turn 4*, as the exercise concludes.

5.6. Some software-related challenges

Using Questionmark’s jump blocks to achieve appropriate frame-based dialogue control is not without its challenges. Careful testing is needed to ensure that the transitions to follow-up questions occur as intended, and that feedback phrases (some of which are associated with particular student responses, while others result from a combination of responses), complement one another when they occur side by side, conveying a

coherent message overall. Strictly, these are the concerns of the developer of natural language (NL) dialogue systems, and software specifically designed for NL dialogue system development will sometimes provide GUI tools that represent diagrammatically, as a directed graph, the possibly complex, looping transitions between steps (McTear, 2004, pp. 164 ff.). By contrast, the developer attempting an approximation of a conversational user interface in Questionmark must be prepared to deal with linear, text-based views of the various question blocks and the conditions that cause transitions between them. Ultimately, this limits the complexity of the dialogue-like interaction that the developer might attempt to realise using such a tool. But to some degree the awkwardness of the authoring tool is compensated by the precision of the interaction (the student ticks boxes that unambiguously convey his/her responses, and the system reacts accordingly).

It is likely that, eventually, other technologies will make for much more naturalistic tutoring dialogues. Google Assistant (<https://developers.google.com/actions>) is one recent technology with distinct potential – though, in that particular case, one of the challenges for the developer is to restrict the system’s ability to interpret the student’s answers (i.e. its in-built natural language ‘understanding’, at least to the extent that it can process (near-)synonyms and paraphrases), so that it accepts *only* the precise answers that the more fastidious tutor really wants it to accept. In everyday conversation the difference between ‘extend’ and ‘extends’ may be minimal, and often might simply be overlooked: in a highly technical environment, however, it is the difference between a reserved word in the UML and a reserved word (with different connotations) in an OO programming language like Java. Future research will examine the suitability of different technologies as tutorial authoring tools.

6. The reaction

6.1. *Who has been using the tutorial?*

The approach has now been trialled with three different second-year cohorts over the past two years, all working to roughly the same syllabus, though with slightly different project themes. As part of a mix of teaching resources (including lectures, advisory sessions, on-line lecture slides and references to recommended textbooks (e.g. Booch et al., 2005 and Dennis et al., 2015), the tutorial provides one more opportunity for students to consolidate their knowledge of use case notation and terminology, and adds another a string to the teacher's bow of teaching resources. On a 'live' module that counts towards their final degree, students can be treated as subjects of an academic 'experiment' only to a very limited degree. With the first two cohorts the use case tutorial was trialled as a revision aid before end-of-course examinations (to check that it worked and to gather feedback: a low-risk strategy since the second-year final examination – also written in Questionmark – usually contains only a few very basic use case questions that are already covered by conventional teaching material). While there is evidence that students who took the tutorial also did better at the relevant questions in the exam, it would be rash to attribute this outcome to the tutorial alone: it is likely that students who already have a good exam technique also make use of all teaching resources offered. Most recently the tutorial was introduced earlier in the second-year teaching programme, where it provided a reminder of some key points just as the students embarked on a substantial team-based project, whose first deliverable included (as with such projects previously) a use case requirements analysis. A comparison of this *most recent* cohort's average score (69.5%) with the *previous* and *last-but-one* cohorts' scores for a similar exercise (both 65%) reveals a modest improvement in performance – a percentage difference of 4.5%. But again, many

factors – not just the tutorial – are at play here, including variations in ability between the cohorts, as well as the lecturer’s growing ability to anticipate and address potential difficulties in an increasingly tried-and-tested package of lecture material. As it continues to evolve technically, the automated tutorial is expected to grow in importance as a means of delivering new and additional course content. Accordingly, more rigorous experiments, including pre- and post-test instruments, will be developed to measure its particular effect on learning outcomes.

6.2. Structured student feedback

Of greater significance at present is the students’ very encouraging feedback on the automated tutorial. A Questionmark survey of the largest cohort to whom the tutorial was made available (212 students, 47 of whom completed the survey) yielded the following outcomes:

- 35/47 (75%) thought the tutorial was very helpful or quite helpful;
- 45/47 (96%) expressed a strong or moderately strong wish that such tutorials should be longer; and
- 45/47 (96%) said that they certainly or probably would like more tutorials like the one they had just had.

These outcomes are represented graphically in Figures 7 – 9.

Figure 7. Did you find the tutorial on requirements modelling helpful?

Figure 8. Would you like this kind of interactive tutorial to be longer?

Figure 9. Would you like more interactive tutorials like the one on Requirements Modelling?

6.3 Free-text student comments

Free-text comments taken more generally from the second-year Questionmark surveys 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.”
- “[It] Was very helpful and the feedback was detailed.”
- “It made me think about the direction the [<<]extend[>>]s and [<<]include[>>]s flow.”

- “It was very practical. It forced me to look at the example and read it to test my knowledge...or lack of. I definitely prefer this more practical approach as sometimes you can fool yourself into thinking that you understand the concept but putting into practice lets you see what learning you need to go over. I loved how I received feedback straight away.”

The main requests from the surveys are for longer automated tutorials with more questions, and for more widespread use of automated tutorials (as one student commented: “Brilliant idea – wish there was one [tutorial] for each topic.”)

6.4. The scope and purpose of the automated tutorial

In its current format, the tutorial is too limited in scope to be used as a standalone primer on use cases: this was not its purpose. In subject-specific terms, it provides a brief but engaging, always-available pep-talk on some use case basics that have already been introduced in class. To do so, it has widened the predominantly one-way interaction between students and standard assessment technology into something resembling a person-to-person dialogue. In particular, the free-text comments in the feedback surveys reveal how well-disposed students are to this automated, dialogue-based approach to learning: this outcome (at a time of challenging student-staff ratios in many institutions and on many courses) is relevant well beyond the field of software engineering.

7. Going beyond the automated tutorial

7.1. Applying use case analysis to project work

Once they embark on their software engineering project, the scope of the students’ work exceeds the simple abstract examples of the tutorial, but the expectation is that the

relationships illustrated in the tutorial will be faithfully applied. Depending on their degree pathway, students on the Software Engineering module have developed either a game (for example, a trading or fantasy card-game using Android and Java), or a web app (e.g. for example, an on-line service, developed in PHP, to identify and reserve a personal tutor), and in each case, early in the process, the students must represent the expected, observable system behaviour in a use case analysis. To take the example of the fantasy card game (one in which players compete by playing virtual cards with different strengths and weaknesses against each other), a competent group will typically identify up to twenty use cases. These will be represented in a use case diagram that features the standard relationships between use cases, the ones rehearsed in the automated tutorial. An edited extract from such a diagram is shown in Figure 10. For example, a *Give battle* use case might <<include>> use cases *Select warrior*, *Select deck* and *Take turn*; use cases *Strike* and *Use warrior power* might <<extend>> *Take turn*; and use cases *Arm weapon* and *Invoke spell* might be specialisations of *Play card* (which itself might also <<extend>> *Take turn*). As the automated tutorial reminds them, most, if not all, students avoid referring to their sets of sequences of actions as

Figure 10. Part of a use case diagram representing behaviour of a fantasy card game.

‘user cases’!

7.2. Finding a suitable form of words...

Beyond the scope of the present tutorial, students are reminded in class and in their lecture slides to make sure that their written use case descriptions correspond closely to the message conveyed by their use case diagram. Students following this advice will, in writing their use case descriptions, devise and apply set phrases that correlate with the relationships shown in their use case diagrams – for example:-

- to flag an extension point: “If [*player chooses an option*] [*Use case name*] use case is called” (thus, “If the player chooses to remove a card from their deck, the *Remove card from deck* use case is called”);
- or to flag an inclusion: “[*Description of step in flow*] ([*Use case name*] is enacted)” (thus, “The player chooses the deck they wish to play with (*Choose deck* use case is enacted)”).

7.3. Building on a good analysis

Having established the functionality that they intend to deliver – and with some leeway to revisit and revise their analysis in the context of a broadly iterative and incremental development process – the best teams go on to produce sophisticated object-models and object-based implementations that impress with the rich functionality they deliver. For second-year students this is invaluable experience for the project-based dissertation they must produce in their final year, usually after a year’s placement in industry. There many will be exposed to – and will be in a position to judge critically – other development methodologies.

8. 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 behaviour and so to the wider development process – provided the student, with the educator’s help, has learnt to avoid common conceptual and notational pitfalls associated with use cases. An automated tutorial can help students feel more comfortable with use case basics, serving as an aid that they can access in their own time and as frequently as they wish, as they prepare for projects or exams. In future, automated tutorials are likely to provide guidance on other aspects of use case analysis and the broader development process, helping students adhere to the good practice that is currently conveyed by more traditional forms of teaching. It is also likely that, as new language technologies mature, more naturalistic automated exchanges will become possible – even though, then, as in any free-flowing conversation, a new challenge will be to keep exchanges natural, but technically precise! The present study has ascertained the feasibility of and students’ openness to automated tutoring. Future work will include controlled experiments to measure the efficiency of different authoring technologies and the effectiveness of the resulting tutorials in achieving improved learning outcomes. While software engineering will remain the context for the research, the approaches developed will be relevant to other subject areas too.

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. At a talk attended by the present author in London in the early 1990s, Jacobson told how use cases had even influenced the layout of his kitchen, the notion being that if a typical task like *Make toast* were a use case, the resources required

(bread, toaster, knife, butter, plate) should all be in close proximity to each other – like a mechanism of collaborating objects, ready for a use case realisation! As with so many great ideas, the power of use cases lies in their broad applicability, extending beyond the realm of software engineering and into everyday life. As a means of encouraging up-front thinking, they are to be welcomed.

References

- Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The unified modeling language user guide*. (2nd ed.). Upper Saddle River, NJ: Addison Wesley.
- Cohn, M. (2017, December 8). Advantages of the “As a user, I want” user story template. Retrieved from <https://www.mountangoatsoftware.com/blog/advantages-of-the-as-a-user-i-want-user-story-template>
- Dennis, A., Wixom, B.H., & Tegarden, D. (2015). *Systems analysis & design: an object-oriented approach with UML* (5th ed.). Hoboken, NJ: Wiley.
- Firesmith, D. (2004). Generating complete, unambiguous, and verifiable requirements from stories, scenarios, and use cases. *Journal of Object Technology*, 3(10), 27–40.
- Fuentes, J. M., García, A. I., Ramírez-Gómez, Á., & Ayuga, F. (2014, March). Computer-based tools for the assessment of learning processes in higher education: a comparative analysis. In L. Gómez Chova, A. López Martínez, & I. Candel Torres (Eds.), *IATED 2014. Proceedings of the 8th international technology, education and development conference*, Valencia, Spain (pp. 976–984). Valencia, Spain: IATED.
- Jacobson, I., Christerson, M., Jonsson, P., & Övergaard, G. (1992) *Object-oriented software engineering: a use case driven approach*. Wokingham: ACM Press/Addison-Wesley.
- Jacobson, I., Spence, I., & Bittner, K (2011). *Use-case 2.0: The guide to succeeding with use cases* [PDF ebook]. Retrieved from https://www.ivarjacobson.com/sites/default/files/field_iji_file/article/use-case_2_0_jan11.pdf
- Jacobson, I., Spence, I., & Kerr, B. (2016, April). Use case 2.0: The hub of software development,” *ACM Queue*, 4(1), 94–122.

- Laguna, M. A., & Marqués, J.M. (2008, July). On the multiplicity semantics of the extend relationship in use case models. In J. Cordeiro, B. Shishkov, A. Ranchordas, & M. Helfert (Eds.), *Proceedings of ICSOFT 2008. Third International Conference on Software and Data Technologies*, Porto, Portugal (pp. 62–75). Heidelberg: Springer.
- McTear, M. F. (2004). *Spoken dialogue technology: Toward the conversational user interface*. London: Springer-Verlag.
- Meyer, B. (2014). *Agile! The good, the hype and the ugly*. Cham, Switzerland: Springer International Publishing.
- Meyer, B (2015, February 18). *Agile methods: The good, the hype and the ugly* [ACM Webinar] Retrieved from <https://www.youtube.com/watch?v=ffkIQRq-m34>
- Mountain Goat Software (2017, December 8). User Stories. Retrieved from <https://www.mountaingoatsoftware.com/agile/user-stories#section-3>
- O'Neill, I. (2018, March). Using assessment software to create a dialogue-based tutorial. *ACM Inroads*, 9(1), 38–44.
- O'Neill, I., Hanna, P., Liu, X., Greer, D, & McTear, M. F. (2005). Implementing advanced spoken dialogue management in Java. *Science of Computer Programming*, 54(1), 99–124.
- O'Neill, I., & McTear M. F. (2000). Object-oriented modelling of spoken language dialogue systems. *Natural Language Engineering*, 6(3-4), pp 341–362.
- O'Neill, I. (2017, September). In praise of use cases. In P. J. Clarke & A. Pierantonio (Eds. – Educators Symposium Chairs), *EduSymp 2017, 13th Educators Symposium at Models 2017, Symposium conducted at the ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems*, Austin, TX (pp. 529–532). Retrieved from <http://ceur-ws.org/Vol-2019>.
- Oran, A.C., Nascimento, E., Santos, G., and Conte, T. (2017, September). Analysing requirements communication using use case specification and user stories. In J.C. Maldonado (Ed.) *SBES '17. Proceedings of the 31st Brazilian Symposium on Software Engineering* (pp. 214–223). Fortaleza, CE, Brazil: ACM.
- Podeswa, H. (2010). *UML for the IT business analyst* (Chap. 6, 2nd ed.). Boston, MA: Course Technology.
- Pressey, S. L. (1926). A simple apparatus which gives tests and scores – and teaches. *School and Society*, 23(586), 373–376.

- Rus, V., D'Mello, S., Hu, X., & Graesser, A. C. (2013, Fall). Recent advances in conversational intelligent tutoring systems. *AI Magazine*, 34(3), 42–54.
Retrieved from <https://aaai.org/ojs/index.php/aimagazine/>
- Śmiałek, M., Jarzębowski, N., & Nowakowski, W. (2012, September). Runtime semantics of use case stories. In M. Erwig, G. Stapleton, G. Costagliola (Eds.), *VL/HCC 2012. Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 159–162). Innsbruck, Austria: IEEE Computer Society.
- Stikkolorum, D. R., Demuth, B., Zaytsev, V., Boulanger, F., & Gray, J. (2014, September). The MOOC hype: can we ignore it? Reflections on the current use of massive open online courses in software modeling education. In B. Demuth and D. R. Stikkolorum (Eds.), *MODELS educators symposium, EduSymp 2014* (pp. 75–86). Symposium conducted at the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, Valencia, Spain. Retrieved from <http://ceur-ws.org/Vol-1346>.
- Suppes, P., Jerman, M. and Brian, D. (1968). *Computer-assisted instruction: The 1965–66 Stanford Arithmetic Program*. New York, NY: Academic Press.
- Wautelet, Y., Heng, S., Hintea, D., Kolp, M., & Poelmans, S. (2016, November). Bridging user story sets with the use case model. In S. Link, J.C. Trujillo (Eds.), *Advances in conceptual modelling. Proceedings of the ER 2016 workshops* (pp. 127–138). Gifu, Japan: Springer.
- Wiegers, K., & Beatty, J. (2013). *Software requirements* (Chap. 8, 3rd ed.). Redmond, WA: Microsoft Press.

Acknowledgement: The Version of Record of this manuscript has been published and is available in Computer Science Education, Published Online 17th May 2018
<http://tandfonline.com/10.1080/08993408.2018.1472949>