Semantic and Structural Difficulties with the Unified Modeling Language Use-Case Notation version 1.3

By Karl Cox and Keith Phalp
Empirical Software Engineering Research Group,
School of Design, Engineering and Computing, Bournemouth University,
Talbot Campus, Fern Barrow, Poole, Dorset, BH12 5BB
Email: {coxk, kphalp}@bournemouth.ac.uk

A case study was undertaken to examine and apply the UML use-case notation version 1.3. This study shows that the notation is open to interpretation and the semantics of the use-case relationships are confusing. The attempt to bolt on the object-oriented inheritance structure to the use-case notation is shown to cause problems with users. Generalisation tries to give use cases an object-oriented feel despite the fact that use cases are fundamentally orthogonal to object-orientation [1].

The case study undertaken involved students developing a use-case diagram and descriptions for a proposed application. We describe a general explanation of the findings. The first finding is that there are risks in making use cases too generic. For example, in this study two of the use cases were combined into one by many of the students. This unification makes some sense, as the two functions appear to be the same except for what appeared to be small details. However, the requirements are very different: one is to request new equipment; the other is to move existing equipment to another room or computer. One could argue that the overall requirement would be to install equipment (hardware and/or software) but the effects on the problem domain would be different.

The second finding was a concern that arose over the communication association lines. That is, how does one show a passive actor receiving some information? For example, a use case "Reply to Requests" shows that a "Manager User" actor instantiates that use case. It does not show which actors are passive, awaiting a response from that use case. To do that would mean drawing a line from the "Reply to Requests" use case to the awaiting actors. This does not seem a problem, except that to do so would imply that the passive actor would also be capable of instantiating the "Reply to Requests" use case, which should not be the case. Solving this problem is relatively simple: one either documents this or, better still, changes the notation to include arrowheads on the communication association lines. The UML version 1.1 indicates direction by showing arrowheads. We question why version 1.3 has disposed of the arrowheads since we found the use-case notation more understandable with them.

The third finding is that use cases often share phenomena with other use cases. These shared phenomena might be, for example, user name, equipment type and location for installation etc. It is difficult to see how such shared phenomena can be described in a use-case diagram without modelling internal communication or interactions between use cases. But Jacobson states: "Internal communication among use-case instances must not be modelled," [2]. It is unclear whether Jacobson means:
i. A part of a use case communicates with another part of a use case, or
ii. How the internals of one use case communicate, or
iii. Whether Jacobson is referring to communication between use cases in the system because use cases are internal to the system.
Writing exceptions/alternatives in the use-case descriptions goes a long to way to contradicting the first option. Use-case descriptions can contradict the second option. Use-case relationships contradict the third possibility. What really needs to be modelled is the fact that use cases can share phenomena (unless they are intended to be exclusive descriptions).

The fourth finding was that students interpreted use-case generalisation to be the same as object-oriented generalisation. Only if generalised use cases were classes in an OOA/D diagram, would there be a case for an abstract superclass, and subclasses that inherit the data and operations of the superclass. However, use-case generalisation states that general use cases can be instantiated. Is there a case when they cannot? This gave the students the wrong impression that all superclasses are instantiable in an OOA/D diagram. We believe that it would be less better to keep to <<include>> relationships rather than using generalisation.

The fifth finding was misinterpretation of the <<include>> and <<extend>> relationships. The rules state that included and extended use cases cannot be instantiated directly by an actor. However, in this study it was clearly intuitive to show an <<include>> use case directly instantiated by an actor. The same situation occurred for an <<extend>> use case.

The format for the use-case descriptions developed in this case study is structured English because it is commonly used in industry [3]. The use-case descriptions give a vast amount of information, much more than can be understood from the diagram. The sixth finding was that use-case descriptions are concerned with specification rather than requirements, something that has been pointed out before [4].

The seventh finding showed the diagram did not depict all the use-case relationships described in the descriptions; to do so would have cluttered the diagram, making it less readable. The UML's focus on the semantics of the diagram's notation is apparent. Project problems, or even failure, due to a lack of concern with the semantics and structure of use-case descriptions is too high a price to pay.

To sum up, we find that the UML semantics that concern themselves with use-case relationships are counter-intuitive in practice. The fact that the UML wishes to make use cases object-oriented, as implied by the generalisation relationship, draws into question the purpose of a use case and also at what level of abstraction to place the diagram and descriptions. If a use case is too generic, the more misunderstandings and misconceptions this will have on the problem domain. How to deal with shared phenomena must be a concern for future versions of the use-case notation - the current notation is inadequately equipped to deal with this issue. Use-case descriptions describe the actor-system interface. However, the UML does not particularly concern itself the structure and semantics of description [5]. We believe that use-case descriptions hold the key to what really needs to be done to get the system right and the UML should now concern itself with this fundamental issue.

References

[1] Donald G. Firesmith (1995), Use Cases: The Pros and Cons, Report on Object Analysis and Design, volume 2, issue 2, pp2-6.
[2] Ivar Jacobson (1994), Basic Use-Case Modelling (continued), Report on Object Analysis and Design, volume 1, issue 3 (September-October), pp7-9.
[3] Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke and Peter Haumer (1998), Scenarios in System Development: Current Practice, IEEE Software, March/April, pp34-45.
[4] Michael A Jackson (1998), A Discipline of Description, Requirements Engineering Journal, volume 3, pp73-78.
[5] Grady Booch, James Rumbaugh and Ivar Jacobson (1999), The Unified Modeling Language Use Guide, Addison Wesley.