Four Roads to Use Case Discovery
There Is a Use (and a Case) for Each One

Gary A. Ham
Battelle Memorial Institute

Use case-based requirements definition is a hot topic, particularly in object-oriented software engineering circles. Appropriate content is achieved by looking at potential use cases from four different views. Each view provides unique advantages. Together they offer the information needed to develop the fully elaborated use cases that facilitate clearly defined, understandable, measurable, and testable design.

Requirements analysis includes both the gathering of functional and system requirements and the organization of those requirements into a logical, traceable, and understandable form. It is one of the most discussed and least well-implemented parts of the software engineering process. As a result, poor requirements analysis is a leading cause of failure in systems development [1]. To address this situation, use case-based requirements definition is becoming popular for systems analysis in general and object-oriented development in particular.

Although use cases are well accepted in principal, the form a use case should take, the level of granularity it should encompass, and even the specific definition of the term "use case" are still matters of dispute in the industry. As a result, most Department of Defense (DoD) contracting officials still prefer to see traditional structured methods and good old-fashioned "shall" statements for requirements definition. This article introduces how to "find" use cases and what it takes to elaborate use cases into effective tools for user validation, operational metrics, and system design. Interestingly, use case can be implemented without throwing away the value of the traditional shall statements and without tossing mission-based structured decomposition out the window.

Use Case Definition

A use case is a sequence of events, performed through a system, that yields an observable result of value for a particular actor.[1] The key issue for requirements management in this definition is the words "observable result of value." The primary goal of requirements definition should be the provision of value. Because use cases, by definition, fit that goal, they are used as the primary organizational structure for requirements definition. The additional components needed of a fully elaborated use case are

- Actors that collaborate in the use case.
- Events (and associated business rules) in which the actors collaborate.
- Information that is passed and returned in the course of each collaboration.
- Context (environment) in which the use case takes place.

Context can best be defined in terms of additional requirements that affect the use case in terms of inputs, controls, outputs, and mechanisms.[2] Input descriptions are requirements associated with what input is available and in what form it can or should be provided. Controls impose algorithmic restrictions on how and when the use case must be performed by prescribing rule sets and regulations that are mandatory. Output descriptions add specific formatting and content requirements to the basic use case product. Finally, mechanism prescriptions are associated with architectural requirements in the sense of logical interfaces to current or planned systems. Enabling collaborations with actors beyond the primary actor that will or must interface in the prescribed use case are also identified.[3]

Four Ways to Create Use Cases

There are essentially four ways to create use cases in sufficient detail to be included in formal requirements in a form suitable to generate implementable systems design and test specifications:

- Mission decomposition – a form of traditional hierarchical structured analysis.
- Unstructured aggregation – collect and classify traditional shall statements.
- Scenario story-driven discovery – use cases are discovered by analyzing written descriptions of day-to-day activity or desired activity.
- Actor or responsibility discovery – first define the actors and roles, then define their collaborations and responsibilities.

Mission decomposition begins with a particular mission goal. The goal must be a clear statement. It may not, in and of itself, have clear metrics for its achievement. If so, the goal must be decomposed into components in a fashion analogous to use of the goal, question, metric (GQM) paradigm that is often advocated to discover software metrics [3]. Beginning with mission defined in terms of a goal, question what accomplishments (products, services, etc.) are required to reach the goal. Decomposition continues until all of the lowest-level accomplishments can be described in terms of a measurable result for a specific user in support of the top-level mission. In other words, decomposition continues until each "leaf node" accomplishment contains the basic output specification for a use case. These output specifications then become the definition around which use case elaboration takes place. Elaboration includes identifying the events, actors, information structures, business rules, and non-functional requirements that apply to the particular mission component.

Unstructured aggregation is used to collect and classify requirements collected from various venues in the form of shall statements and business rules.[4] Any active voice shall statement that describes an individually measurable product or service that must be provided for a particular actor becomes a candidate use case. All other requirements are reviewed for their applicability to the use cases discovered. Generally, these additional requirements are applicable in a descriptive sense as input, output, control, and mechanism requirements depending on how they will affect the further development of the candidate use cases.

A scenario story is a detailed description of all the interactions by one or more users with the system in a set of related events. The story should include details that describe user interaction with the system in a detailed, concretely specified and verifiable form. The scenario story is used for both requirements elicitation and user validation. When written properly, scenario story paragraphs form potential use cases, and sentences within those paragraphs describe the events involved in performing the use case.

Actor, responsibility, and collaboration discovery is a traditional object-oriented analysis technique that begins with finding roles that actors play, what responsibilities they have for task accomplishment, and what other actors they must collaborate with to accomplish those tasks.[5] Use cases are discovered by identifying productive task results. Subtasks leading to those results become events within an identified use case.

So, which approach to use case discovery is best? Each approach offers advantages. GQM-based mission decomposition offers measurable results and a focus on mission rather than fluff and “nice to have.” Shall statements allow formal integration of nonfunctional and architectural concerns into analysis and provide specific reference to requirements, e.g., performance response times, that may apply to multiple use cases. Scenario stories offer a completeness of detail and an effective user validation viewpoint that is difficult to achieve with other approaches. Scenarios are also of great value in obtaining eventual user acceptance of new or changing systems. Finally, no matter which use cases are identified, they cannot be put together until the actors and collaborators in events are identified.

Each approach also has limitations. It is sometimes difficult to obtain mission focus from untrained subject matter experts, even in facilitated workshops. Merely getting consensus on the mission can be an interesting task in some environments. It is much easier to ask, “What do you do each day?” Theoretically, shall statements are individually verifiable and can be clearly written, at least in the microsense. However, because of their “atomic” nature, this is usually not the case. Most of the time, shall statements are poorly organized, ambiguously stated, and difficult to implement or test. They are often redundant and overlapping, yet designers often find large gaps when basic modules are built. Scenarios tend to focus on current process and change based on current process. This makes it harder to think outside of the box. Beginning with scenarios also tends to add requirements that benefit particular users rather than benefit the mission to be accomplished—fluff happens. Finally, the purely bottom-up actor and responsibility approach raises completeness questions and a concern that generated use cases might reflect individuals’ requirements ahead of organizational mission needs. There also are questions about the effectiveness of role and class abstraction in a bottom-up “find the nouns” type of environment.[6]

Since each approach has both benefits and limitations, the question of where to start becomes one of basic expediency. Start with whichever entry point offers the most initial return in information. You can begin with what is most comfortable for the organization under analysis or for the team doing the analysis. You can also reuse existing documentation. If initial scenario stories are available, use them. If prior business process reengineering work has left clear mission descriptions, or defined organizational role and responsibilities definitions, use them. If all you have are large documents filled with poorly structured (or well structured) shall statements, use them, too. The rest of the analysis information can be added at any point, as long as the use case structure to which it is added remains consistent.

Use cases are not requirements in and of themselves. Instead, use cases provide a showcase in which requirements are precisely organized and illustrated for user validation, system design, and test script development. To be effective, a use case needs the following:

- A measurable contribution to a defined mission in support of a primary actor.
- A clear definition of input, output, control, and mechanism-related requirements and business rules.
- A presentation format that facilitates functional user validation and change elicitation.
- An understandable presentation of roles and collaborations by event in a sequence as a basis to assign and find class operations.

Each of the above needs is best served by a different one of the four approaches. So, achieving a high level of effectiveness implies that all four approaches are eventually needed for complete analysis. Leaving one out will reduce the value of that use case as system design or test script development documentation. As long as a defined process to maintain traceability and coordination between approaches is maintained, the particular initial approach is not material. The measure of success will be the clearly defined, understandable, testable designs that result from fully elaborated use cases.

Disclaimer and Acknowledgments

The views expressed in this article are my own (as the author) and do not formally represent those of the DoD or Battelle Memorial Institute. They represent my distillation of collective team member experience in support of the Computer-Based Patient Record Interoperability using Object-Oriented Technology project for the Office of the Assistant Secretary of Defense for Health Affairs. Csaba Eghazy, Scott Eyestone, Carol Fogelsong, Don Heim, and Janet

AMC CSS Achieves CMM Level 3

The Air Mobility Command (AMC) Computer Systems Squadron (CSS), Scott Air Force Base, Ill. received a Level 3 rating during a Software Engineering Institute (SEI) Capability Maturity Model (CMM) assessment. The CSS currently has over 450 employees dedicated to developing, maintaining, and enhancing transportation and command and control software systems for AMC. The assessment culminated 17 months of dedicated hard work.

One requirement to achieve Level 3 was to develop and maintain a usable set of software process assets that improve performance across all projects and provide a basis for cumulative, long-term organizational benefit. They developed a process asset library (PAL) located on the Web at http://cpssweb.safb.af.mil:81/pal/pal_home.htm. The AMC CSS PAL is accessible to everyone within the military and government Internet domains.

Martino provided valuable insight that is reflected in some form in this article. ◆

About the Author

Gary A. Ham is a senior research scientist for Battelle Memorial Institute, National Security Division, Information Systems Engineering and Process Modernization Department in Arlington, Va. A former Marine Corps comptroller and Naval Academy computer science instructor, he currently researches value metrics definition processes to support object-oriented requirements analysis and design of DoD systems. He has a bachelor's degree in economics from Whitman College in Walla Walla, Wash. and a master's degree (with distinction) in information systems management from the Naval Postgraduate School in Monterey, Calif. He is currently a doctoral candidate in information technology at George Mason University in Fairfax, Va.

Principal Research Scientist
Battelle Memorial Institute
2101 Wilson Blvd., Suite 800
Arlington, VA 22201-3008
Voice: 703-575-2118
Fax: 703-671-9180
E-mail: ham@battelle.org

References

1. Research Report, "Chaos," The Standish Group, 1995, http://www.standishgroup.com/chaos.html.
2. Jacobson, Ivar, Martin Griss, and Patrick Jonsson, *Software Reuse: Architecture, Process, and Organization for Business Success*, ACM Press, New York, N.Y., 1997.
3. Fenton, Norman E., *Software Metrics, a Rigorous Approach*, Chapman and Hall, London, 1994.

Notes

1. Ivar Jacobson's basic definition differs slightly: "A use case is a sequence of transactions performed by a system, which yields an observable result of value for a particular actor" [2]. For our purposes, an actor is defined as a participant in a use case event, as an instigator, a provider of service or product, or as a recipient of that service or product.
2. If this sounds a little like Integration Definition for Function Modeling (IDEF0), it should. IDEF0, with a difference in focus from functional decomposition to product or service identification, can effectively be used to identify mission-focused use cases. The required change in mindset may be difficult for traditional IDEF0 modelers. It was for me. If you can make the transition, however, a whole new approach to software metrics based on activity-based costing becomes available.
3. The particular form that a use case should take is less important than the content. The only requirement is a consistent presentation of use case contents that provides clear understandability by subject matter experts. The use of formal notation languages, e.g., Unified Modeling Language and predicate logic, should be left out unless the user community is fully conversant in the notation presented. We use a standard format for our use cases. This "standard" has, however, been adjusted in each analysis iteration to better meet the understandability needs of our validating users.
4. Business rules are defined to be requirements that contain a conditional phrase, e.g., "if," or "then." Business rules are designed to govern the actions of an event or events, either singly or grouped in a rule base. In some references, nonconditional rules are referred to as business rules. My current project merely calls such rules requirements. We feel the distinction is important because business rule sets can be used within rule engines to process events depending on condition. Straight requirements act regardless of condition.
5. Although analogous, this is not the same as the class, responsibility, and collaboration approach. We are defining roles that will probably (but may not) be assigned to classes as part of the design process. We do not try for class definition in analysis use case development. We save that for design, when architectural dependency issues are more completely specified.
6. Yet, we have used this approach extensively for project management. All of our task statement development and project work breakdown structures are based on the definition of responsibilities and collaborations between project teams where project teams are recognized as actors or classes in the object-oriented sense. Project management is object management to the extent that Gantt charts are defined by a composition of sequence diagrams developed from the original collaboration diagrams.