2022

Synergistically Employing User Stories and Use Cases in the Practice and Teaching of Systems Analysis and Design

Gary Spurrier
University of Alabama - Tuscaloosa, gspurrier@cba.ua.edu

Heikki Topi
Bentley University, htopi@bentley.edu

Communications of the Association for Information Systems

Accepted Manuscript

Synergistically Employing User Stories and Use Cases in the Practice and Teaching of Systems Analysis and Design

Gary Spurrier
Department of Information Systems, Statistics, and Management Science,
The University of Alabama
gspurrier@cba.ua.edu

Heikki Topi
Department of Computer Information Systems,
Bentley University

Please cite this article as: Spurrier, Gary; Topi, Heikki: Synergistically Employing User Stories and Use Cases in the Practice and Teaching of Systems Analysis and Design, *Communications of the Association for Information Systems* (forthcoming), In Press.

Communications of the Association for Information Systems

Research Paper ISSN: 1529-3181

Synergistically Employing User Stories and Use Cases in the Practice and Teaching of Systems Analysis and Design

Gary Spurrier
Department of Information Systems, Statistics, and Management Science,
The University of Alabama
gspurrier@cba.ua.edu

Heikki Topi
Department of Computer Information Systems,
Bentley University

Abstract:

Over the past three decades, user stories and use cases have become increasingly dominant requirements techniques. Both support articulating functional requirements for software projects, although they evolved within different software development approaches—user stories from agile development and use cases from traditional software engineering—and differ significantly in the level of requirements detail they can capture. As such, user stories and use cases are neither synonyms nor mutually exclusive alternatives. Rather, they can and should be *complementary* in the systems requirements process. Unfortunately, this mix of similarities and differences—coupled with a lack of formal standards for either—make understanding and synergistically employing user stories with use cases confusing and challenging for practitioners and students alike. To address this, this paper first provides a descriptive overview of the evolution of user stories, use cases, and their interrelationship. Second, it fills a gap in the literature by providing a prescriptive, detailed approach to employing user stories and use cases together. This prescriptive approach is illustrated via a comprehensive tutorial example, providing practitioners with actionable skills and SA&D teachers and students with a new pedagogical tool.

Keywords: Requirements specification, user stories, use cases, IS education.

[Department statements, if appropriate, will be added by the editors. Teaching cases and panel reports will have a statement, which is also added by the editors.]

[Note: this page has no footnotes.]

This manuscript underwent [editorial/peer] review. It was received xx/xx/20xx and was with the authors for XX months for XX revisions. [firstname lastname] served as Associate Editor.] **or** The Associate Editor chose to remain anonymous.]

1 Introduction

Since the early 1990s, user stories and use cases have become increasingly important and widely used methods for articulating functional requirements for software projects. They have tended to displace older, alternative requirements approaches. Assessed from a management information systems (MIS) discipline perspective, this is evidenced by that most leading MIS Systems Analysis and Design (SA&D) textbooks have taken to referring to other, older techniques such as data flow diagrams as "traditional" requirements approaches (Dennis, Wixom, & Tegarden, 2015, pp. 7, 24, 112, 127, 184, 218, 264; Satzinger, Jackson, & Burd, 2016, pp. OL-20-OL-38) or distinguishing the "traditional" structured systems development life cycle (SDLC) approaches from the more modern agile and object-oriented approaches that utilize user stories and use cases, respectively (Kendall & Kendall, 2014, pp. 28-32 for use cases and pp. 165-8 for user stories; Tilley & Rosenblatt, 2017, pp. 400-1; Valacich & George, 2017, pp. 15-23, 183-4).

Given these points and the foundational importance of requirements analysis to system projects in general and to systems analysis and design in particular, we expect that MIS practitioners and SA&D students—who are, after all, budding practitioners—would need a deep, integrated understanding of user stories and use cases. More specifically, we expect that they would need a complete, clear, and actionable understanding of user stories and use cases in terms of the following issues:

- Concisely defining both user stories and use cases.
- Comparing their similarities and differences, including key goals, techniques, and benefits of each.
- Explaining if their use should be mutually exclusive or complementary, including actionable guidance regarding:
    - The circumstances in which each should be used.
    - How and when they should be utilized together.

Note that these are not abstract or academic concerns, given the diversity of software development approaches used in organizations today. As an example, a major, global, multi-industry survey of agile software development practices (Digital.ai, 2020) found that there is approximately a 50/50 split between organizations in which less than half of teams are agile vs. those in which more than half of teams are agile. While the exact meaning of "agile" in this context may be debated, overall, this points to that software development organizations today continue to employ a wide range of approaches. To use terminology introduced by Boehm and Turner (2004, pp. 37-39), this includes a wide diversity of requirements management and delivery approaches, ranging from:

- **Agile methods,** such as Extreme Programming and Scrum, which express requirements scope as a backlog of informal, evolving, and prioritized **user stories**.
- **Plan-driven methods,** such as the traditional SDLC, which express requirements scope in terms of formal, complete specifications, such as may be found in detailed **use cases**.
- **Hybrid methods,** which may combine detailed up-front requirements similar to plan-driven methods with iterative software construction similar to agile methods.

How does the MIS discipline make sense of these questions and points? Again, leading SA&D textbooks can serve as a reasonable proxy for the content taught in typical SA&D MIS courses. But even a cursory analysis of these books reveals a considerable lack of consistency in explaining the interrelationship of user stories and use cases (Spurrier & Topi, 2018). Some books seem to focus on user stories, while others emphasize use cases, and the majority do not provide an integrated treatment of the two.

Given this inconsistency, graduates of MIS programs may enter the job market with an incomplete and inconsistent understanding of these topics. IT professionals arriving in the field from other disciplines may face even greater challenges in achieving a solid, integrated understanding.

In this paper, we address these topics in a systematic, well-sourced manner, pursuing two objectives:

- **Descriptive history and understanding:** Enabling the reader to make sense of the origins of user stories, use cases, as well as their interrelationship in terms of their intertwined history and parallel evolution. This will help the reader overcome several sources of confusion that complicate understanding of "how we got here." These sources of confusion include:
    - **Shared roots but divergent evolution:** User stories were actually inspired by use cases, but then evolved in a different direction based on a radically different set of values.

  - **Multi-year evolution:** Both user stories and use cases have evolved significantly over the years and from the work of multiple authors and commentators. Because of this no single, seminal work comprehensively explains all of the concepts and tactics of either technique.
  - **Lack of consistent standards:** In general, neither user stories nor the most valuable aspects of use cases—use case *narratives*—are codified via a formal standard, such as the Object Management Group's Unified Modeling Language (OMG's UML). Because of this, terminology differences between different authors and commentators have tended to persist. Further, specifically with respect to use cases, the UML does specify a standard, but only for use case *diagrams*, which offer less value and are less frequently used than use case narratives.
- **Prescriptive method for complementary use:** From the prior objective, we identify a key gap in the field pertaining to specifying how to synergistically employ user stories with use cases in a detailed, actionable manner. The basic idea is that teams utilizing user stories to define project scope may in certain circumstances find themselves in need of a way to elaborate those user stories in a systematic and fairly formal way. It is this "user stories first, sometimes followed by use cases" linkage that has caused us to articulate this article as "user stories and use cases," rather than the other way around. (Note: below, as we explain the origins and evolution of user stories and use cases, we do reverse that articulation when we recognize that use cases were invented first and then inspired the idea of user stories.) We highlight how authors and other contributors from both the agile and plan-driven camps at times endorse linking user stories and use cases in this complementary fashion. Further, however, they generally stop short of providing specifics as to how that linkage should be implemented. The reason this gap seems to have persisted is because each commentator tends to speak primarily from the perspective of either agile, emphasizing user stories, or plan-driven, emphasizing use cases—but none seem to have directly bridged that gap at a detailed level. We explicitly fill that gap by proposing and illustrating a new, systematic method for linking and structuring work based on user stories associated with use cases. This linking approach leverages analogous concepts within each technique. We conclude with a specific, comprehensive tutorial example illustrating this integration approach.

Based on this review and tutorial, IT practitioners will achieve a practical, actionable understanding of how to effectively utilize these techniques in a complementary, synergistic fashion, and SA&D instructors and their students will acquire a corresponding pedagogical resource for improving the teaching of this subject.

2 The Evolution of User Stories and Use Cases

2.1 The Issue: Practitioners Trying to Make Sense of Their Options

Note that, per the discussion above, we expect that most readers will not have a full, robustly-sourced prior understanding of user stories, use cases, and their interrelationship—indeed, providing that understanding is a key objective of this paper. However, in order to gain the most benefit from the paper, the reader will need some general level of familiarity with one or both of these techniques. This is because both user stories and use cases are vast topics, with numerous lengthy books having been written about each. As such, a single article cannot provide complete coverage of these topics.

We can, however, summarize and provide original sources for the key characteristics of user stories and use cases. As a starting point, we offer a short history of user stories and use cases. This will help explain and resolve the sources of confusion noted in the introduction and set the stage for the prescriptive approach for integrating the two techniques in the second half of the article.

We first review the general ideas and characteristics of user stories and use cases as a practitioner or student today would typically encounter them through SA&D textbooks or practitioner-oriented trade books. We highlight a sometimes-confusing potpourri of user story and use case concepts and practices that are often, but not always, found together and seldom explained consistently.

This will set the stage for "understanding how we got here" via a high-level but comprehensive historical review of the origins of those concepts and practices. This will be based on seminal articles and books from key authors and other contributors in the agile and plan-driven spaces.

Via this review, we identify and explore potential sources of confusion arising from:

- Common roots for these techniques that diverged widely as they were developed in support of agile and plan-driven software development approaches and values.
- An ongoing evolution of each technique from multiple authors, leading to non-standard, fragmented technique definitions and specifications.
- A lack of recognized, formal standards for the most valuable aspects of both techniques, exacerbating the prior point with non-standard terminology.

Via this discussion, we provide a detailed analysis of the differences, similarities, and interrelationships between user stories and use cases, helping the reader achieve a comprehensive and integrated understanding of the two techniques.

2.2 Typical Practitioner Experiences Today: User Stories vs. Use Cases

As noted above, SA&D textbooks are inconsistent in their handling of user stories and use cases, such that the understanding of the budding practitioners (the students completing these courses) is also likely to be inconsistent. The experience of practitioners entering an information system requirements role from another educational discipline is likely to be even more confusing. There are literally hundreds of trade books available that address these topics, and there are no standards or other clear guidance as to which of these would offer a comprehensive, authoritative coverage of the material.

Much of a practitioner's understanding may therefore emerge informally from observing IT team practices, reading a book or two, and reading Internet blog entries. From that experience, we suggest that the following ideas—and, at times, confusion—regarding user stories and use cases may emerge.

2.2.1 User Stories

What does a "user story" mean to a typical practitioner today? While we will flesh out the sources of these points in detail below, based on the contents of recent, popular trade books available today, we would anticipate that a typical practitioner would describe a user story as including many of the following ideas, in approximately a descending order of likelihood as shown below—a user story may be:

- Captured as a functional user feature—what the system will do to support a type of user or role—described in a brief, non-detailed manner. Typically captured on an index card, sticky note, or equivalent electronic format in one of many available agile planning systems (of which Jira may be the best-known example).
- Expressed using the "role-feature-reason" format: "As a <type of user>, I want <system feature> so that I can <achieve goal or obtain benefit>." There may be a sense that this is a rigid formula, or there may be some sense that departures from the format are sometimes allowed.
- Utilized in agile methods such as eXtreme Programming or (more frequently today) Scrum, although nowadays often seen in other methods, as well.
- Grouped with other user stories, which are rank ordered by priority to form a project's scope in the form of a "product backlog."
- Grouped into (or split out from) an aggregate user story called an "epic" or a "theme," although a precise definition of and distinction between these may be elusive. Motivations for doing this may include:
    - Sizing a story to fit into a fixed iteration of work (a "sprint").
    - Enabling differentiation between higher and lower priority item work.
- Understood to be a "promise for a conversation" or, in an alternative formulation, part of a "card, conversation, and confirmation" ideal. Both of these ideas point to an avoidance of creating detailed, up-front, comprehensive requirements in favor of determining requirements via frequent, on-going, informal conversations with business customers.
- In connection with "confirmation" from the prior point, user stories are clarified with additional "acceptance criteria," "conditions of satisfaction," and/or "acceptance tests," although the exact nature and format of these and their similarities/differences may be unclear. For example, are these three terms synonyms or different concepts (or may they be small user stories, themselves)? In terms of their meaning, are they functional requirements? Non-functional requirements? Risks, issues, assumptions, or constraints? Some or all of these? Is there a specific format for their expression?

- Evaluated for quality via something called INVEST criteria: A story should be Independent, Negotiable, Valuable, Estimable, Small, and Testable, although the exact meaning of these may be hazy.

Figure 1 provides an example of the role/feature/reason user story format and also of a specific user story from a familiar application context—placing an order for items in a shopping cart at a retail webstore. One can easily imagine this as being one of numerous of user stories needed to describe the application requirements, including other user stories such as searching for items, placing the items in the cart, and so on.

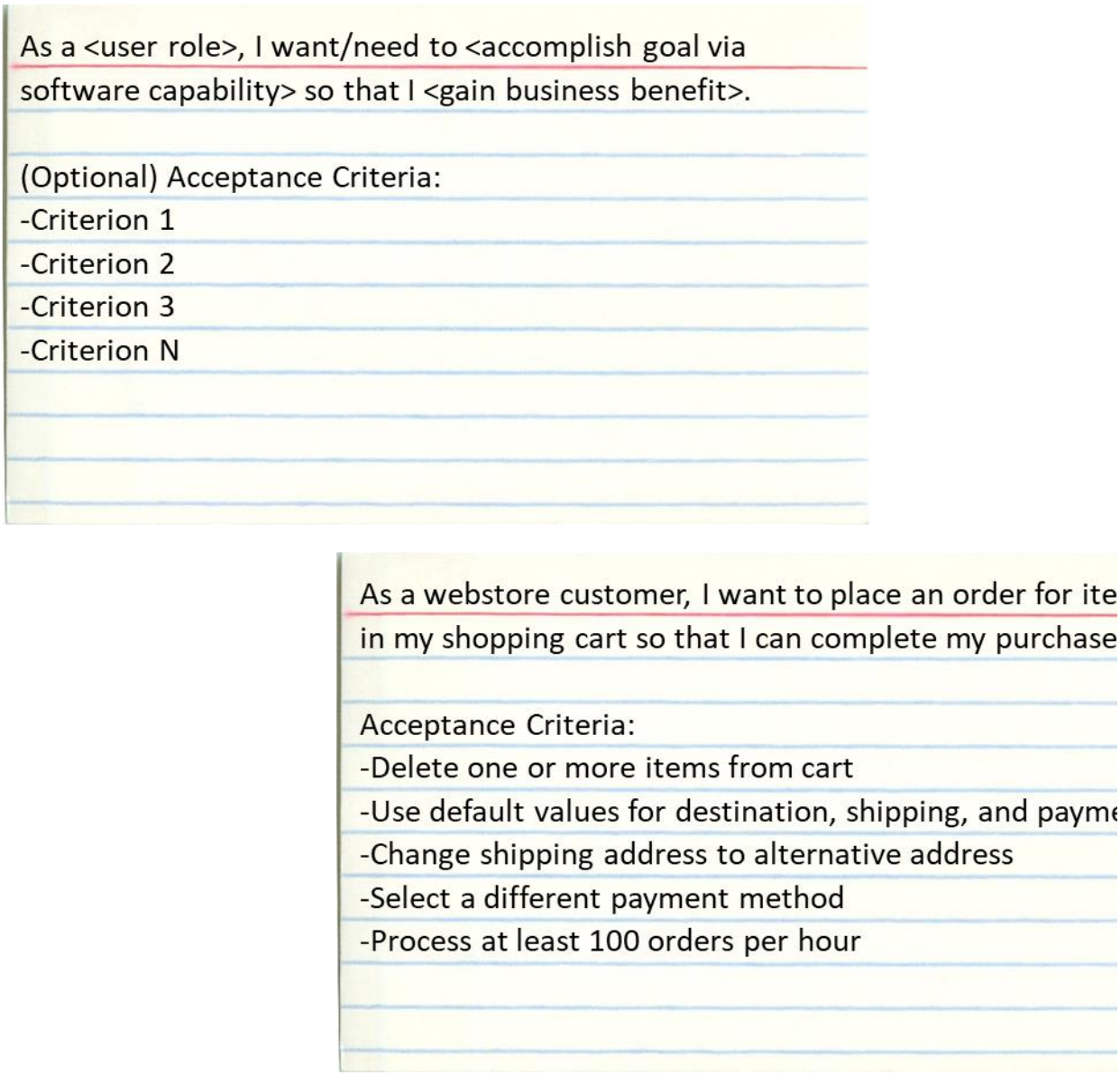

Figure 1. User Story Format and Example

We suggest that most practitioners would mention only a subset of these points in their description of user stories and would express varying levels of confidence as to their meaning and importance.

2.2.2 Use Cases

What does a “use case” mean to a typical practitioner today? As with the corresponding user story section above, we will flesh out the sources of these points in more detail below, but we would anticipate the following general understanding—a use case may be:

- A formal, sometimes (but not always) detailed description of functional requirements in the form of the steps a type of user takes to accomplish some goal with a system. Less detailed versions may resemble user stories, and, for some teams, short use cases may be used in place of user stories.
- Captured as a “stick figures and ovals” diagram or as text or both, although the reasons for doing so, the relative value, and the interrelationship of these different expressions may be hazy.
- Comprised of a variety of sections, especially in more detailed expressions. Multiple formats exist, and it will often not be clear which is the “right one.”
- Utilized in a traditional SDLC or any other methodology in which a team determines that detailed, up-front functional requirements are important to capture.
- Together with other use cases, may comprise the complete specification of functional requirements for a systems project.
- An expression of either an entire user experience in achieving a goal with a system or else a single task within that goal.
- Including different “scenarios,” “flows” or “extensions” that capture what should happen with variations, exceptions, or error conditions.
- A way to tie together other types of requirements, such as a domain model, user interface model, non-functional requirements such as performance, security, and so on.
- A source of test cases for a QA team.
- When user stories are also being used, used to elaborate details for a user story, although whether that should be done for all or a subset of user stories may not be clear.
- The scenarios/flows/extensions noted above can be used to:
    - Size work to fit into software construction iterations or sprints.
    - Manage scope by adding or removing support for specific variations and exception handling processes.

Figure 2 provides an example of a highly detailed use case narrative or description (both terms are used by different authors) that corresponds to the user story in Figure 1. It utilizes a format and contents suggested by noted use case authority Alistair Cockburn (2001). Importantly, the Main Success Scenario describes the simplest, end-to-end version of the use case—sometimes called a “happy path”—while the Extensions modify the Main Success Scenario with variations and exceptions/errors.

Note in this example that the Extensions in Figure 2 correspond to the Acceptance Criteria in Figure 1—this is intentional on our part and will figure into our prescriptive recommendations later for using user stories and use cases in a synergistic manner.

Figure 3 provides a use case diagram utilizing the Unified Modeling Standard “stick figures and ovals” format. This example also corresponds to the same system context as Figures 1 and 2—note that each combination of actor and use case can be seen as being generally similar to the base definition of a use case narrative at a very high level of abstraction. Thus, the combination of the Webstore Customer actor and the Place Order use case corresponds to the user story in Figure 1 and the use case narrative in Figure 2. Use case diagrams can be useful for articulating a broad system context, but they are outside the scope of this discussion.

As with user stories, we suggest that most practitioners would mention only a subset of these points and, within a given point, would use inconsistent interpretations and terminology.

2.2.3 User Stories and Use Cases: Perceived Similarities and Differences

Scanning the bullet points and the examples in the two sections above, we see many similarities but also some key differences in user stories and use cases. In Table 1 below, we array these characteristics side-by-side to illustrate how a user may be confused by this mixture of similarities and differences. We express these in the same generally descending order of likelihood of being mentioned as we used above.

Use Case Narrative

Use Case Name: Place a Webstore Order

User Story: As a Webstore customer, I want to place an order for items in my shopping cart so that I can complete my purchase.

Stakeholders and Interests: Webstore customer, warehouse, accounts receivable

Related Models and Mockups:

File Name and Location	Model/Mockup Name
RetailSiteDomainModel.png	Retail Site Domain Model
ShoppingCartMockup.png	Screen mockup of shopping cart

Preconditions: Customer must be registered and one or more items must exist in the customer's shopping cart.

Minimal Guarantee: Errors are logged, customer is not charged, and cart items are not updated to "Ordered" or shipped.

Success Guarantee: Cart items are updated to "Ordered," customer is charged the correct amount, and warehouse is notified.

Main Success Scenario:

1. Customer accesses their shopping cart.
2. System displays customer's shopping cart, including a list of the items placed in the cart.
3. Customer indicates that she/he wants to place an order for the items in the cart.
4. System displays a new order, including all the items in the cart with the quantities for each as specified in the cart, and blank values for destination, shipping method, and payment method.
5. Customer specifies destination, shipping method, and payment method, then indicates order is complete.
6. System computes and displays the total amount for the items in the order, shipping costs given the chosen shipping method, sales tax, and total cost of the order.
7. Customer places the order with the selected values.
8. System displays and e-mails customer an order confirmation message and sends order to the warehouse.

Sample Extensions

3a. Customer decides to delete one or more items from cart.
- 3a1. Customer selects items to delete from the cart.
- 3a2. System updates cart, retaining any items not marked for deletion.
  - 3a2a. No more items remain in the cart.
    - 3a2a1. System deletes cart and use case ends.
- 3a3. Customer indicates she/he wants to place an order for the remaining items.

5a. Customer decides to use default values for destination, shipping, and payment.
- 5a1. Customer indicates she/he wants to order using default values for destination, shipping method, and payment method.
- 5a2. System fills in those default values from the customer's profile.
- 5a3. Customer indicates order is complete.

5b. Customer decides to change shipping address to an alternative address.
- 5b1. Customer indicates she/he wants to use an alternative address.
- 5b2. System displays previously stored addresses and an option to add a new address.
- 5b3. Customer chooses between selecting an existing address and adding a new shipping address.
- 5b4. System saves chosen address.

5c. Customer decides to select a different payment method.
- 5c1. Customer indicates that she/he wants to use a different payment method.
- 5c2. System displays previously stored payment methods and an option to add a new payment method.
- 5c3. Customer chooses between selecting an existing payment method and adding a new payment method.
- 5c4. System saves selected payment method.

Figure 2. Textual Use Case Example Following Structure by Cockburn (2001)

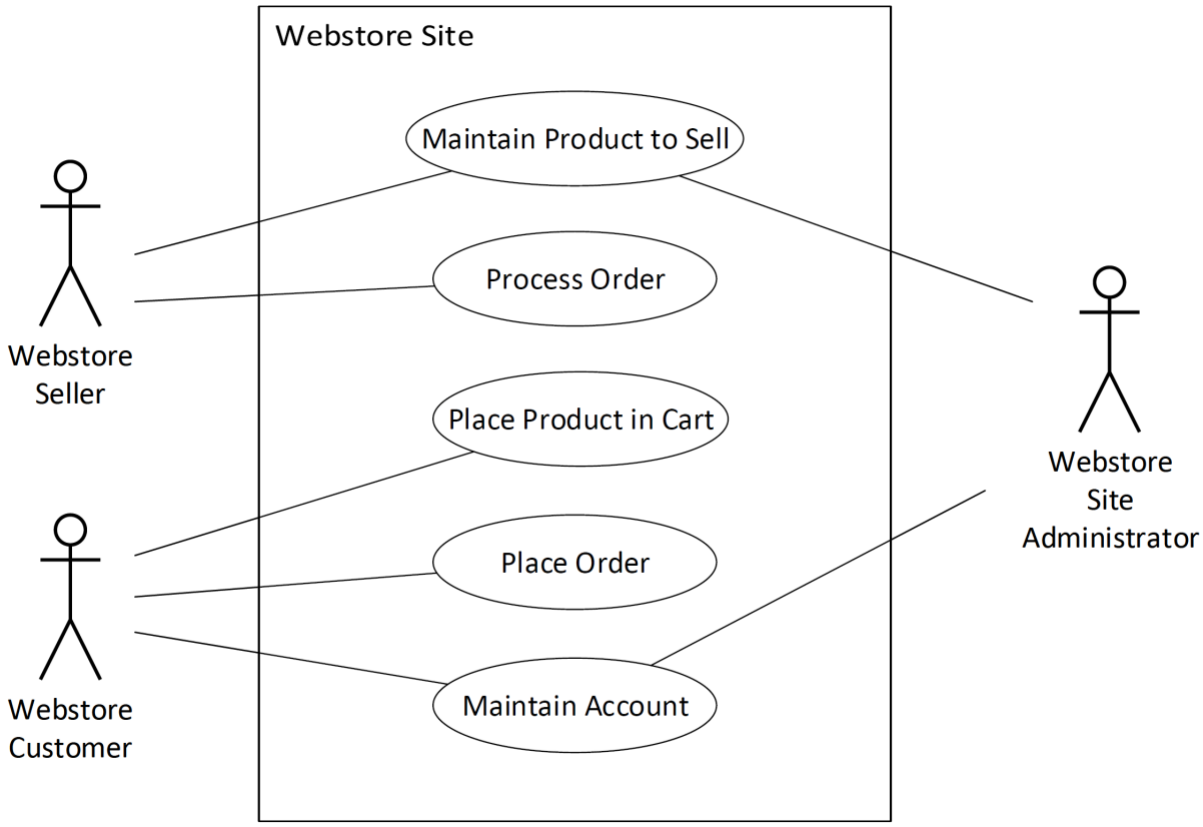

Figure 3. Example of UML standard Use Case Diagram

Considering Table 1, we can clearly see that for many of the issues describing user stories and use cases, the two techniques appear to have shared purposes for existence:

- Provide functional feature requirements definitions, each of which describes a system feature meeting a user need
- In their briefest forms are expressed at a similar level of summarization
- Provide mechanisms for establishing test cases
- May be expressed as large, complex features containing a series of smaller, simpler features (and vice-versa)

On the other hand, differences seem to exist, as well:

- The level of detail in a use case narrative may be far greater than that in a user story—indeed, the user story merely provides a summary description of a functional feature that sets the stage for an informal conversation with a system customer (or "product owner," in Scrum terminology) to fully understand the detailed requirements. This is in keeping with agile's opposition to complete, formal specifications. The use case, in contrast, can explicitly and formally expand that summary description to any required level of detail.
- There seems to be a difference in scope definition. User stories are associated with delivering flexible scope via prioritizing user stories into agile product backlogs, often by rank ordering the stories. In contrast, use cases are most frequently associated with defining the fixed functional requirements scope in the traditional SDLC, other plan-driven methods, and many hybrid approaches.

Table 1. Practitioner Understanding of User Stories vs. Use Cases

Descriptor	A user story may be:	A use case may be:
Thumbnail definition	Functional user feature—user, feature, and value—expressed briefly as text and recorded on index card, sticky note, or similar electronic format	Functional user feature expressed as a user-goal combination, either as a summary diagram or textually as a series of interaction steps
Expression format	Who/what/why of a feature: "As a <type of user>, I want <system feature> so that I can <achieve goal or obtain benefit"	Varies from a formal UML "stick figures and ovals" diagram to a comprehensive structured textual narrative
Utilized in	Agile methods (XP, Scrum, etc.)	Traditional SDLC or other approaches, including hybrid, that specify detailed, up-front requirements
Project scope	Ranked ordered with other user stories to form a "product backlog"	Comprises system functional requirements, but may be prioritized using MoSCoW criteria or similar approach
Abstract/ concrete	"Epics" or "themes" are large user stories that can be split into smaller user stories	"Summary-level" use cases incorporate multiple "user goal level" cases, which in turn may incorporate multiple "subfunction" use cases
Detail	A "promise of a conversation" rather than detailed requirements	Level of expression varies widely: -Briefest resemble user stories -Most detailed may be several pages long May incorporate or reference many other types of requirements (domain, UI, etc.)
Testing	Acceptance criteria, "conditions of satisfaction, or acceptance tests	Main flows used to source test cases for QA team
Evaluated by	INVEST criteria	

From this discussion, we more clearly see how user stories and use cases are at once highly similar yet also clearly distinct in their nature and intended use in software development projects. In and of itself, this is not an earth-shattering observation. However, again stepping into the shoes of an IT practitioner engaged in requirements analysis and design, we confront several key questions:

- Is my understanding of user stories and use cases complete and correct? What am I missing or misinterpreting? Where did all these ideas come from?
- Are these fundamentally different concepts, such that I should only deal with them independently, or are they closely related in a complementary manner, so that it makes sense to use them together synergistically?
- To the extent I see this landscape completely and correctly, what should I do with that?
    - Should I use user stories when utilizing an agile methodology? What should I do when I have to adapt an agile methodology with more detailed up-front requirements—e.g., because the project is large, complex, is in a regulated industry requiring detailed requirements, the team is diverse and far-flung, etc.?

- Should I utilize use cases because they seem more inherently adaptable (e.g., they can accommodate a wider range of detail than user stories can)? If I decide to utilize use cases, should I create each one at the same level of detail?
- Should I utilize user stories and use cases together? If so, how do I relate one to the other? Should the approach be highly consistent, or should I adapt it to each feature (and, if so, how and on what basis)?

2.3 A Brief Evolutionary History of Use Cases and User Stories

We will further explore the implications of Table 1 later. But first, we begin to answer the questions above by first solidifying our understanding of user stories and use cases via a brief history explaining the sources and evolution of each technique. Note that above we have used the formulation "user stories and use cases"—this is because, in terms of SA&D requirements, we are envisioning a process in which user stories are identified first and then are elaborated via use cases.

In this section, however, we reverse the order of the nouns: "use cases and user stories." This is because, historically, use cases came first, giving rise to user stories as a sub-category—albeit a very different sub-category—of use cases.

Note that all the characteristics noted in Table 1 and the bullet points above are accounted for below. Throughout this discussion, we reference Figure 4, which provides a summary timeline of seminal publications from both techniques.

Also, to help the reader navigate differing terminologies encountered in use cases, we provide a table of corresponding terms in Table 2. In this table, we highlight in bold the terms that we utilize in the discussions and examples in the final sections of the paper.

- **1986-Introduction of Scrum:** Takeuchi and Nonaka publish "The New New Product Development Game" in *Harvard Business* Review (Takeuchi & Nonaka, 1986), which introduces key ideas of agile product development in Scrum—albeit in a general context not specific to software development. Note that there was no discussion of user stories initially in Scrum.
- **1987-Introduction of use cases:** Ivar Jacobson, the father of use cases, publishes the first article introducing the concept of a use case, defined as a "…sequence of transactions, perform by a user and a system in a dialogue…" and "…described…by using structured English or…a data flow chart…" (Jacobson, 1987). Use cases are the main tool for specifying a system's functionality (Jacobson et al, 1992, p. 125). This was presented in the context of the burgeoning topic of object-oriented design, which was then emerging as a major alternative to structured techniques. However, then, as now, use cases, themselves, focus on specifying functionality, which is then allocated to objects (classes) in a separate model (Jacobson et al., 1992, p. 137-8).
- **1992-Wider adoption of use cases:** Jacobson and several co-authors publish *Object-Oriented Software Engineering: A Use-Case Driven Approach* (Jacobson et al., 1992), which includes now-familiar elements of use cases:
  - Asserts that use cases are the main tool for defining the functional requirements of a system (p. 125).
  - Contrasts two versions of use case expression (pp. 156-7, 188-9):
    - Summary diagrams (i.e., ovals and stick figures).
    - "Prose-like form" textual descriptions.
  - Use of "basic course" (read: basic flow or main success scenario) to designate the most fundamental version of the use case that should be described first and "alternative courses" (read: alternative flows or extensions) to designate variations and exceptions (p. 157).
  - Notes that the basic and alternative courses (read: flows) should be a source of test cases (pp. 327-8).
- **1995-Scrum described for software development:** By Ken Schwaber (1995). Note that this version does not mention user stories, but, rather, defines scope in terms of "backlog items" (a term still used as a near-synonym for user stories in Scrum today)

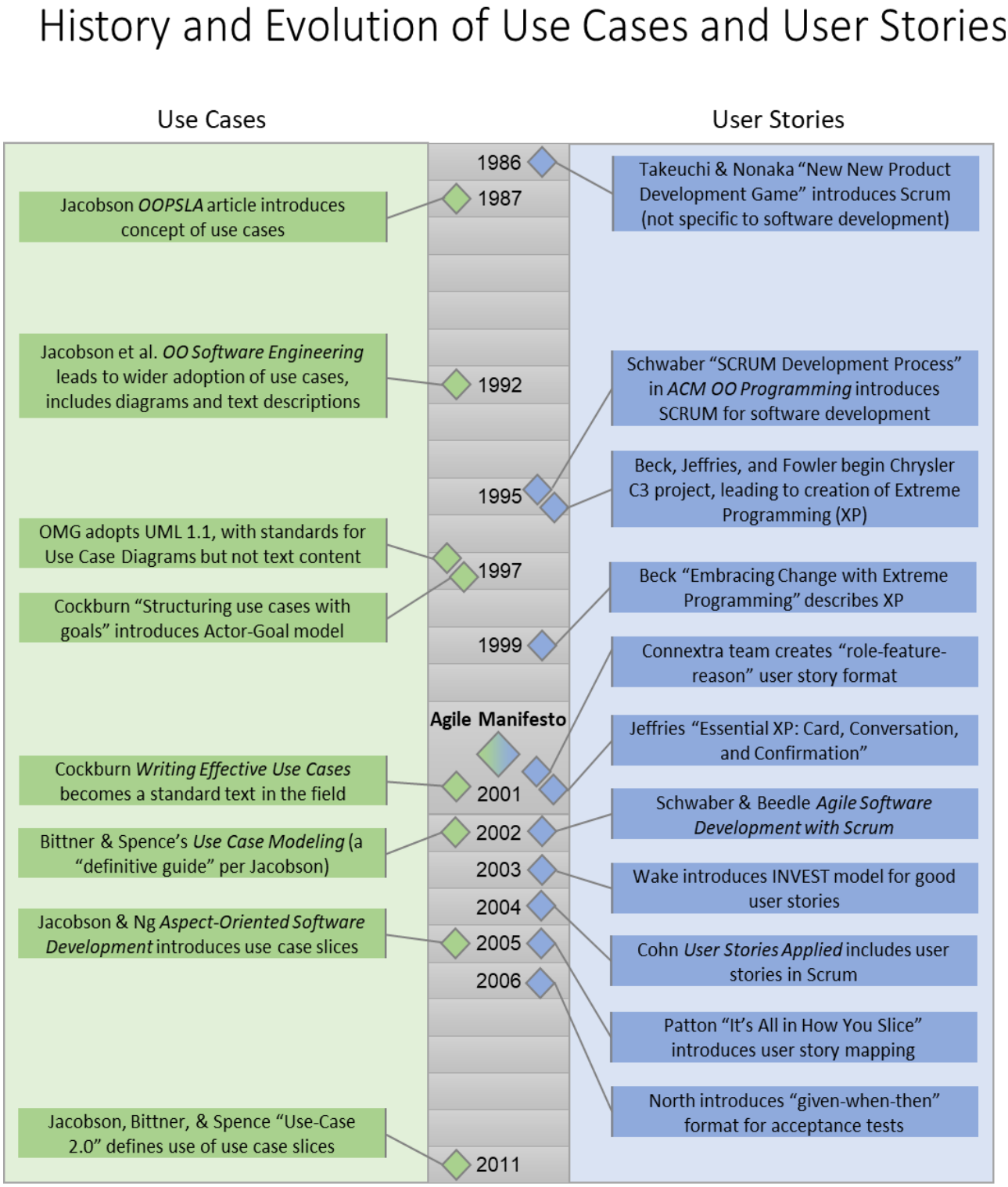

Figure 4. History and Evolution of Use Cases and User Stories

Table 2. Use Case Terminology

Core concept	Cockburn (2001)	Jacobson, Bittner, & Spence (2011)
Briefly stated format (similar to a user story)	Use Case Brief (p. 120-1)	Briefly Described (p. 47)
Highly elaborated, fully-detailed format	**Fully Dressed Use Case** (pp. 119-20)	Fully Described (p. 48)
Basic flow of steps without variations, exceptions, or errors	**Main Success Scenario** (pp. 87-8)	Basic Flow (p. 7)
Steps supporting variations, exceptions, or errors	**Extension** (pp. 99-110)	Alternative Flow (p. 7)
Use Case segmented (split or sliced) into specific subsets of the basic flow and other steps, often to provide chunks of functionality buildable in a single iteration or release	Complete Scenarios (pp.170)	**Use-Case Slice** (pp. 8, 15, 31-2)

- **1995-Beginnings of eXtreme Programming:** Kent Beck, Ron Jeffries, and Martin Fowler begin work on the "C3" project for Chrysler Corporation. This project was the crucible that led to the creation of the eXtreme Programming (XP) agile method of software development, including the introduction of user stories.
- **1997-Object Management Group adopts UML 1.1, including Use Case Diagrams:** As a diagram-centric standard, OMG only set standards in UML for the use case "ovals and stick figures" diagrams. They did not provide standards for the textual contents of each use case. That the UML is silent on use case textual contents has led to a high degree of variability in use case contents and approaches (Fowler, 2004).
- **1999-eXtreme Programming introduced:** Kent Beck publishes seminal article on eXtreme Programming (XP) (Beck, 1999). Key points include:
    - "Stories" are functional features (the term "user stories" was introduced later). Beck notes that the inspiration for stories was Jacobson's work on use cases. Defines a story as a short use case: "…you can think of [a story] as the amount of a use case that will fit on an index card." (Beck, 1999, p. 71)
    - Stories here are not written using the now-familiar format role/feature/reason format: "As a <type of user>, I want <system feature> so that I can <achieve goal or obtain benefit."
    - In conjunction with delivering small releases via iterative development, advocated for splitting large stories (based on identifying multiple story parts with different levels of importance).
    - Customers write the "functional tests" for each story.
    - Indicates that XP is the "best way to deliver software in environments where requirements change violently" (p. 73), from which we may infer that XP (and, presumably, other agile methods) may not be the optimal approach in all circumstances, including specifically when requirements are clear and stable.
- **2001-Agile Manifesto published:** Notably includes key XP contributors—Kent Beck, Ron Jeffries, and Martin Fowler—and key Scrum contributors—Ken Schwaber and Mike Beedle—but also includes one of the main voices for use cases: Alistair Cockburn (Beck et al., 2001).
- **2001-"role/feature/reason" user story format created:** By a team at Connextra in the UK, sometimes credited to Rachel Davies (Cohn, 2004; Patton, 2005). Over time this becomes a de facto "standard" format for writing user stories.
- **2001-XP user story "Card, Conversation, Confirmation":** Article by Ron Jeffries (2001) emphasizes that user stories should be written on a card not to document detailed requirements, but rather to stimulate conversations between customers and programmers in which those detailed requirements are determined informally. The needed functionality is confirmed through acceptance tests defined by customers and implemented by programmers.
- **2001-*Writing Effective Use Cases:***
    - Written by Alistair Cockburn (Cockburn, 2001), becomes a standard reference (Fowler, 2004). Cockburn was a student of Jacobson and indicates that he and Jacobson have "no significant incompatibilities" (Cockburn, 2001, p. xx-xxi).
    - However, Cockburn's terminology differs significantly from that of Jacobson. For example, Jacobson's "basic flow" and "alternative flows" correspond to Cockburn's "main success

scenario" and "extensions," respectively. In the absence of a formal standard, this leads to conflicting terminologies.
    - Cockburn supplies terms describing varying levels of descriptive detail in use cases: a "brief" use case may be as short as a user story, a casual use case consists of perhaps two paragraphs, and a "fully dressed" use case consists of multiple sections, including the main success scenario and extensions from the prior point (note that Figure 2 is a fully dressed use case).
    - Cockburn also notes that use cases can be created at several "levels." The most important is the "user-goal" level, describing a user/system interaction accomplished by one primary actor in a single sitting or session. Above that is the "summary-level," involve multiple user-goal sessions or steps. Below is the "subfunction" level, which may be used to describe steps within a user goal level use case.
    - Cockburn explicitly recognizes Extreme Programming and user stories, noting the similarity of user stories and brief use cases. Similar to Ron Jeffries in the prior point, he notes the approach in eXtreme Programming of using user stories as starting point—"promissory notes"—for additional requirements conversations with customers. However, he goes on to argue that the conditions needed for XP to work—direct, close collaboration between customers and IT team members—are frequently not met, suggesting that user stories be extended via use cases.
- **2001-Constraint story cards:** Newkirk and Martin (2001) introduce "constraint story cards" to contain non-functional acceptance criteria. This implies that it is functional acceptance criteria that should be included on story cards.
- **2002-*Use Case Modeling:*** Written by Jacobson collaborators Kurt Bittner and Ian Spence (2002), this book becomes another key source of guidance for writing use cases (endorsed by Jacobson, 2011). Utilizing similar concepts but different terminology than Cockburn's book, it advocates for:
    - Using alternative flows to manage scope, including prioritization mechanisms such as the MoSCoW technique (using categories "must have," "should have," "could have, and "won't have"; Clegg & Barker, 1994). This begins to move use cases away from fixed scope definitions and toward the flexible scope approaches associated with agile iterative development.
    - Advocates for using activity diagrams or flow charts to visualize the overall flow of events as a map to the underlying use cases, a technique similar to agile user story mapping (see Patton, 2005, below).
- **2002-*Agile Software Development with Scrum*:** Schwaber and Beedle (2002) publish definitive guide to Scrum for software development. Note that this still refers to "product backlog items" rather than user stories.
- **2003-INVEST model of good user stories:** Bill Wake (2003) publishes an influential blog entry, arguing that user stories should be Independent, Negotiable, Valuable, Estimable, Small, and Testable. This has become a widely cited ideal for evaluating the "goodness" of user stories.
- **2003-User stories in Scrum:** Mike Cohn (2003) notes that user stories can be used effectively in Scrum, adding user stories to the existing Scrum concept of product backlog items (PBIs), which include functional features (like user stories), but also tasks to deal with defects, technical improvements, knowledge acquisition, and so on.
- **2004-*User Stories Applied*:**
    - The "user stories in Scrum" argument was expanded in Cohn's *User Stories Applied* (2004).
    - Also, includes the concept of adding user acceptance criteria pertaining to variations and exceptions—read: functional acceptance criteria—to the back of a user story card.
    - Argues that large ("epic") stories come in two categories: compound stories that can be easily split into multiple shorter stories and single stories that are inherently complex.
    - Draws parallels between user stories and use cases, including that a user story may correspond to a single scenario in a use case.
- **2005-Introduction of use case slices:** Jacobson and Ng (2005) introduce the idea of use case slices, which are end-to-end paths through a use case representing different paths and incorporating the basic flow (main success scenario) and then alternative flows (extensions).
- **2005-Introduction of user story mapping:** Jeff Patton (2005) argues that the "flat user story backlog"—an unstructured collection of user stories—is difficult to understand, to validate as being complete, and to manage in large products with many stories. He argues for arranging user stories

horizontally in chronological order of their execution in a business process, with higher criticality, "big" (or epic) stories at the top and lower priority stories (e.g., variations) shown below.

- **2006-"Given-when-then" acceptance test format:** Daniel North (2006) creates Behaviour Driven Development, which includes the idea of structuring acceptance criteria using a "given-when-then" structure. This is one of few examples of providing specific guidance for how to write acceptance criteria. Note that this originally specifically focused on functional acceptance tests but is sometimes now extended to user story acceptance criteria.
- **2011-"Use-Case 2.0":** This eBook published by Jacobson, Spence, and Bittner (2011) (and as reiterated and extended in similar a 2016 *ACM Queue* article by Jacobson, Spence, and Kerr) more fully develops the idea and practice of use case slices, and explicitly advocates for using use cases/use case slices in conjunction with agile methods such as Scrum and Kanban to both size work to fit into iterative development and to prioritize the delivery of that work.

3 Complementary Employment of User Stories and Use Cases: Motivations, Common Challenges, and Parallel Principles

The history outlined above confirms and provides sources for the origin of the key user story and use case characteristics outlined in Sections 2.2.1 and 2.2.2, respectively. It also confirms and underlines the similarities and differences between user stories and use cases described in Section 2.2.3. As such, we have now largely fulfilled our first objective of providing a "descriptive history and understanding" of user stories and use cases.

However, in and of itself, this history does not address our second objective of providing a detailed, prescriptive method for complementary use of user stories and use cases—one that will provide synergistic value to IT requirements practitioners and clarity for MIS teachers and students. Constructing a conceptual framework for this prescription is the goal of Section 3, while providing a tutorial example illustrating that framework in use is the goal of Section 4.

In this section, we begin by exploring key motivations for linking user stories to use cases. These motivations fundamentally arise from project characteristics in which the assumptions underlying agile are violated, pointing us toward the need to create more formal, detailed requirements models and documentation than a "pure" agile approach would advocate. This includes elaborating user stories with use cases.

Then, as we approach the challenge of creating a prescriptive method for linking user stories to use cases, we will find it useful to realize that the two techniques share a series of common challenges and corresponding, parallel principles for responding to those challenges. We will see that each of these parallels can illuminate our path, helping us craft an element of an overall approach to effectively and systematically linking user stories and use cases in a complementary manner.

3.1 Fundamental Idea: Using Use Cases to Elaborate User Stories

To begin, as noted earlier, user stories and use cases are both ways of expressing functional feature requirements—system behavior that supports a user in achieving some personal goal or realizing some business benefit. In fact, we learned that user stories were inspired by use cases and, hence, may be considered as a specialized type of use case tailored to the values and expectations of agile development.

Further, the most fundamental difference between user stories and use cases is the level of detail that can be captured: user stories are by convention generally limited to a single statement of the feature, sometimes augmented by a series of "acceptance criteria" or "conditions of satisfaction" (essentially synonymous terms; see, e.g., Cohn, 2017).

As noted above, in contrast to user stories, the level of detail in any particular use case may vary significantly. As initially drafted by a systems analyst, use cases may be relatively brief and lacking in detail. To use terminology from Alistair Cockburn (2001, p. 187), these use case "briefs" may resemble user stories in their brevity. As such, these brief use cases may be used to inventory required system features in a manner similar to user stories.

Having said that, a use case may express not only the goal and value of a feature to a type of user, but also provide a much richer, more detailed expression of the requirements. First, one of the essential goals of a use case is to describe the interaction steps between the user and the system required to achieve the user's goal. Second, Cockburn (2001, pp. 14-15) notes that use cases can also serve as a "hub-and-spoke,"

"project-linking structure" for bringing together other key requirements models detailing how a feature should function and be constructed, such as the domain model, user interface model, non-functional requirements, and so on.

As noted above, neither use case narratives nor user stories are part of the Unified Modeling Language (UML) standard. As such, there is no official standard for either. This lack of standardization is particularly acute for use cases, given the relatively greater level and variety of detail that may be captured in them. However, certain sources have proven to be influential in defining de facto standards. Figure 2 shows a typical, detailed use case format, including key sections specified per Cockburn (2001), elaborating on the base user story "As a Webstore customer, I want to place an order for items in my shopping cart so that I can complete my purchase" shown in Figure 1. Cockburn calls use cases elaborated to this level of detail "fully dressed" (Cockburn, 2001, p. 119). In a concrete manner, this example illustrates how a user story can be elaborated to specify additional detail, context, and related requirements artifacts.

The essential insight of this discussion is that user stories and use cases are neither synonyms nor mutually exclusive alternatives. Rather, they are *complementary* in the systems requirements process. Specifically, as software teams capture scope in the form of a backlog of user stories, they then may capture and communicate additional details and context of some or all of those user stories in the more elaborate use case format. We illustrate the general idea of expanding the level of detail in a user story using a highly detailed, fully dressed use cases in Figure 5.

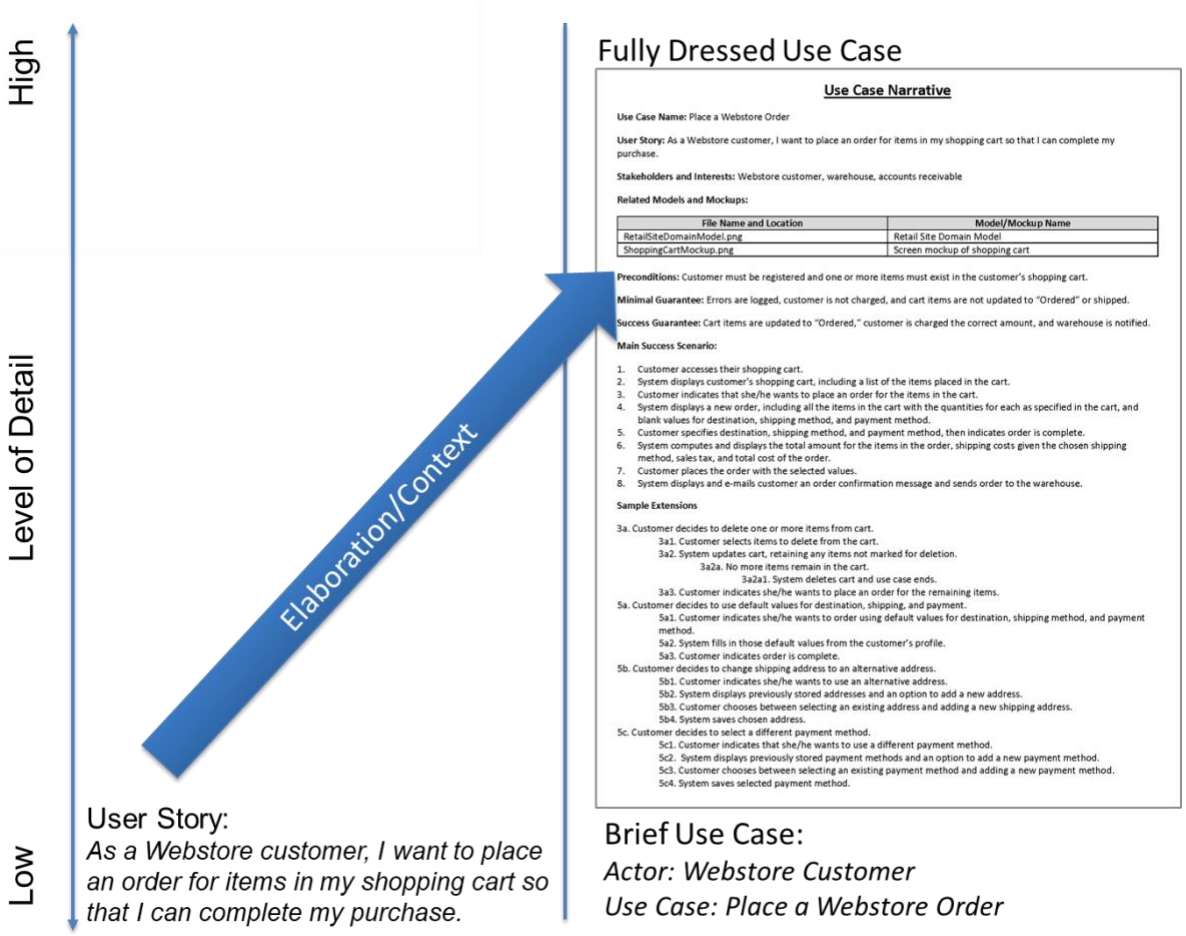

Figure 5. Illustration of Complementing a User Story with a Use Case

3.2 Key Motivations: User Stories and Use Cases as Complementary Techniques

Consistent with the values of the Agile Manifesto (2001), most user story authors discourage the use of detailed, formal requirements documentation, like fully dressed use cases, and one argues that "Use cases were largely banned from the agile tribes" (Leffingwell, 2011, p. 367). So when would deviating from this general guidance be justified?

The need to elaborate user stories with use cases may arise from several different software project characteristics. In general, such project characteristics describe circumstances where key assumptions of agile development are violated, making it necessary to formally document requirements and design details. Such characteristics include:

- **Inability to engage in effective informal, verbal communication:** Commonly arising from a large team—potentially including both business customers and the IT team—operating in multiple locations and/or multiple time zones. But even if the IT and customer teams are small and co-located, this can be a problem if the customer team is not consistently and continually available to the IT team. Within the IT team, this can be a problem when the team members creating requirements must communicate those requirements to a different and, especially, geographically remote group of team members constructing the solution.
- **Large, complex functionality:** For example, highly complicated calculations may require formal documentation of formulas and algorithms to support software development, test case creation, updated user policies, procedures, training documentation, and so on.
- **High probability and impacts of breakage:** Attempting to enhance a large, monolithic, and/or poorly architected application increases the difficulty of successfully modifying a code base. Further, high criticality applications—for example those that are mission critical, contain sensitive customer data, are subject to regulation and audit, or impact human safety—increase the cost of errors and drive the need for more detailed documentation.
- **Legal, regulatory, or contractual requirements:** Regulated industries may require more formal, comprehensive requirements documentation than can be delivered via user stories alone. For outside IT contractors working for such clients, this can be manifested in contractual requirements.

These issues have been noted by both use case-focused authors (Cockburn, 2001, p. 187; Jacobson, Spence, & Kerr, 2016, pp. 120-121) and user story-focused authors (Cohn, 2004, p. 188; Leffingwell, 2011, p. 247). More general frameworks have been developed for assessing project characteristics to strike an optimal balance between agile and plan-driven approaches, including requirements analysis (Boehm & Turner, 2004; Spurrier & Topi, 2017).

In general, these kinds of situations increase systems communications challenges and project risks, thereby motivating the elaboration of a user story with a use case to provide additional written requirements detail. A use case allows the level of detail to be increased to any degree and to include or reference other requirements or design types needed to guide development team members through the details of constructing and testing the solution.

In summary, despite the Agile Manifesto's de-emphasis of "comprehensive documentation," these arguments support the idea that capturing more comprehensive, formal documentation beyond just user stories will in certain circumstances prove worthwhile.

Anticipating arguments below, the prescriptive user-story-to-use-case complementary approach we develop below serves three key requirements management objectives:

1. **Articulating and communicating complex functionality:** In situations where formality is called for, per the discussion above.
2. **Managing scope by *priority*:** By splitting and valuing large functional requirements in a systematic way. Per the discussion in Section 2, agile routinely manages scope by organizing user stories by priority into product backlogs. However, how to manage the priority of requirements *within* a user story (i.e., by acceptance criteria) is less clear. For plan-driven methodologies utilizing more formal requirements such as use cases, Boehm and Turner (2004, p. 154) point out that formal approaches such as traditional software engineering have historically not recognized the need for prioritization, considering each feature to be of equal importance. Our approach leverages and reinforces the use case prioritization techniques advocated by Jacobson and his collaborators (Bittner & Spence, 2003, p. 75; Jacobson, Spence, & Bittner, 2011), extending them to user story acceptance criteria.

3. **Managing scope for *sizing*:** As noted in Section 2, both user stories and use cases include a focus on structuring development work to fit into iterative software construction cycles (sprints). User stories can be split for sizing purposes (Cohn, 2004, p. 32), although we shall see that it can be challenging to determine whether emerging requirements details from one user story should be captured as additional (split out) user stories vs. as acceptance criteria attached to the original user story. In general, as we shall see below, there are many approaches to splitting user stories—likely too many to be useful to students and inexperienced practitioners. Use cases have included the idea of utilizing alternative flows/extensions to prioritize variations and exceptions to the basic functional requirements (Bittner & Spence, pp. 190-192); this idea was later expressed more explicitly as use case slices (Jacobson, Spence, & Bittner, 2011). Our approach leverages use case principles to help discover, structure, and split user stories in a consistent manner. Further, our approach thereby provides a straightforward way to link those user stories to use cases when doing so is worthwhile.

Further, we must answer the following questions:

- How do we structure—that is aggregate or split—both user stories and use cases and the link them in a complementary manner? Is the relationship between user stories and use cases one-to-one, one-to-many, many-to-one, or contingent?
- Which elements of a user story link to corresponding elements in a use case? Specifically, how do we structure each element of a user story—the story itself and its acceptance criteria—and then map them to specific elements of a use case narrative? Are there any situations where elements of a user story that should mapped to other project requirements?

The following sections deal with all these points and questions.

3.3 Challenges of Feature Sizing: Two Key Dimensions of Large Features

We begin now to face the substantive challenges of utilizing these techniques in a complementary manner. While, per Figure 5, it would seem to be a straightforward proposition to take a user story and elaborate its details in a use case, in fact there are some thorny conceptual challenges in doing so.

The first of these arises from the concept of story "largeness"—as noted above, user stories can be conceived of as ranging from small and low-level (concrete) to large and high-level (abstract). Generally, in agile methods such large stories are called "epics" (Cohn, 2004, p. 32). (Note: sometimes the term "theme" is also used for a large story, but sources differ on these definitions and their interrelationships; Rubin, 2013, p. 88. The distinction between an epic and a theme is not central to our discussion, so we do not explore that issue further in this paper.) Because of their size, epics violate the INVEST ideals noted earlier. In particular, epics generally are too large to easily estimate, negotiate, or build in a single iteration. But because they are large, they generally can be split or segmented into smaller stories.

Cohn argues that there are two types of epics (Cohn, 2004, pp. 52ff):

- **Compound epics** can easily be decomposed into smaller user stories.
- **Complex epics** are inherently large and cannot easily be decomposed into smaller user stories.

3.3.1 Compound Epics

We first tackle the issue of *compound* epics. For example, consider the story "As a Webstore customer, I want to shop online." If we recognize this as meaning the entire online retail shopping experience—from arriving at the website to receiving an order confirmation email and all tasks between—then this story is too large and high-level to meet INVEST criteria. Appealing to our earlier example, this overall activity could easily, for example, be broken into several smaller stories, each starting with "As a Webstore customer, I want to…":

- Search for products.
- Place products in a shopping cart.
- Place an order for items in my shopping cart.
- Receive an order confirmation.

Each of these can be recognized as a separate, sequential step or action or task within the larger "shop online" epic, making that a compound epic.

Not all compound stories represent sequential actions. For example, consider a compound epic "As a financial analyst, I want to be able run a suite of financial reports so that I can analyze net income trends." Here, the reporting suite could readily be decomposed into individual reports that do not need to be run in a particular sequence. Decomposing each report as a separate story here would be simple.

However, many compound epics do correspond to sequential, multi-task business processes, as is the case in our webstore example. In these cases, decomposition is more involved: do we have all the right steps? What is their sequence? How do they relate to each other? What are their variations and exceptions? Given this, we make this a key area of focus in our discussion.

Usefully, in the first of several parallels between user stories and use cases, we see that use cases can be categorized in a way that corresponds to the user story concept of a compound epic. Specifically, Cockburn (2001) proposes an approach for categorizing use cases by "levels." A "user goal level" use case represents the interactions of a user and system where the user can accomplish a business goal in a single sitting (session). In many cases, that single sitting represents multiple steps. For example, the compound epic "As a Webstore customer, I want to shop online" could also be modeled as a user goal level use case, with the four bullet points above each representing a step in the use case.

The use case methodology can also be used to describe a business process that consists of several sequential steps involving multiple, related "single sittings." These could involve a single user role in multiple sittings, multiple user roles in a single sitting each, or some combination. Cockburn (2001, p. 64) uses the term "summary level" use case for these types of use cases. Our discussion focuses now, however, on the user goal level use cases, each describing how a user can achieve a "single sitting" user goal. We will discuss this further in our prescriptive approach and tutorial example below.

Returning to the compound epic/user goal level of analysis, we now face the task of making sense of these levels of abstraction. In another example of parallel principles, we note that both user stories and use cases provide broadly similar visual techniques for modeling compound epic/user goal-level step decompositions: user story mapping (Patton, 2005; Patton, 2008) and activity diagrams for visualizing the flow of events in use cases (Bittner & Spence, 2002, pp. 193-4), respectively.

Usefully, variations and exceptions within each step can be modeled within that use case via extensions. This will come into play below, as we examine the handling of story complexity established by acceptance criteria.

Further, addressing the cardinality issue raised above, here the compound epic maps to a single goal-level use case, while each story decomposed and identified via an activity diagram becomes a step in that goal-level use case.

3.3.2 Complex Epics

The other key factor leading to story "largeness" is *complexity*. Complex stories are inherently large and not easy to decompose into smaller stories. For example, in our current user story example "Place an order for items in my shopping cart," the basic functionality seems straightforward. However, requirements may emerge pertaining to the story that add significant complexity, for example (per Figure 1):

- Delete one or more items from cart.
- Use default values for destination, shipping, and payment.
- Change shipping address to an alternative address.
- Select a different payment method.
- Process at least 100 orders per hour.

In general, each of these items (except the last one, a non-functional requirement, which we deal with below) can be seen as a variation of or exception to the base user story—the same role/feature/reason combination—rather than different user stories dealing with a different feature (e.g., searching for products, placing them in the cart).

How should we treat these items? Specifically, should they be treated as:

- Additional user stories
  or as
- Acceptance criteria (sometimes also called conditions of satisfaction) attached to the original user story.

From the perspective of user stories, the answer can be imprecise. Mike Cohn, for example, has written that either approach is acceptable (Cohn, 2017), but that he suggests writing these items as additional user stories, rather than as acceptance criteria attached to the original user story, based on either or both of the following being true:

- Adding the items as acceptance criteria would make the original story too large to fit into a construction iteration.
- The items are of differing priority.

We identify several potential challenges with this. First, overall, it is an imprecise approach to structuring user stories and, therefore, provides little guidance to practitioners and students. Second, it drives the structure of requirements, in part, based on specific characteristics of the team and its implementation approach (the team's velocity and the length of an iteration). Presumably, if a team switched to shorter sprints during a project, the user stories might need to be revised, with some acceptance criteria becoming separate user stories. Third, given the emphasis on progressive refinement of user stories in agile development, it does not seem to follow that different acceptance criteria must all be of the same priority to be part of the same story. Fourth, this could lead to a profusion of user stories, compounding Jeff Patton's concerns above regarding the difficulty of managing large collections of "context-free" user stories.

More generally, Patton notes his own discomfort with agile's general lack of precision in decomposing user stories (Patton, 2008, p. 197). He uses the metaphor of user stories being like rocks being hit by a mallet: no matter how many times we hit the rocks with mallet, the results are more rocks. It is unclear how acceptance criteria fit into this conception.

To impose useful structure on this, we leverage use case concepts to help us implement a clear decision rule for determining whether these kinds of additional requirements should be handled as separate user stories or as acceptance criteria. Specifically, we utilize the concept of use case "slices" (Jacobson, Bittner, and Spence, 2011). A use case slice defines multiple paths from the beginning of a user case to the achievement of the goal at the end of the use case—in user story terms, corresponding to variations on the same user/feature/value combination. The most basic flow (main success scenario) is the most important and first slice to build. Additional slices can be defined by selecting alternative flows (extensions) to invoke representing variations and exceptions to the main success scenario.

In the example above, we can see that all but the last bullet point represent variations or exceptions to the basic flow of the original user story. As such, we can define and prioritize additional slices (beyond the first, main success scenario slice) and handle them as separate backlog items that can be individually prioritized and estimated. It is worth noting that Jacobson et al. refer to slices a "stories." However, they don't call them "user stories," and it seems doubtful that agile practitioners would consider use case slices as synonyms for agile user stories.

As shown in Figure 2 above, we have taken the specific step "Place an order for the products in the shopping cart" out of the original compound epic "As a Webstore customer, I want to shop online" and elaborated it into its own use case narrative. This would be a "subfunction" use case, to utilize Cockburn's use case level taxonomy, given that it does not alone lead to an achievement of a an overall, single-sitting user goal. Within the use case, the functional items in the bullet points at the beginning of this section show up as extensions to the main success scenario of the use case.

Note that this detail could have been added within a single step in an overall goal-level "As a Webstore customer, I want to shop online" user goal level use case, but we split it out as its own subfunction use case because including this level of detail would make the user goal level use case long and hard to read (Cockburn, 2001, p. 110).

Turning our attention back to user stories, this justifies why, in the corresponding user story in Figure 1, we have recorded these items as functional acceptance criteria, rather than adding them as separate user stories. We can see that those acceptance criteria correspond exactly to the extensions in the corresponding, task level use case narrative in Figure 2. Functional requirements not extending the base user story would instead be captured as separate, additional user stories.

3.4 Challenges of Feature Value: Prioritizing Features in Large-Scale Projects

As noted earlier, agile has significantly moved the field of software development forward with respect to recognizing the importance of the value and, therefore, the priority of software features (Boehm & Turner, 2004, p. 154). This flexible scope approach contrasts with traditional, fixed scope approaches, which

historically frequently treated all requirements as equally important. But even agile techniques utilizing user stories suffer from some limitations. Specifically, while agile, backlog-driven techniques routinely prioritize stories in the backlog via rank ordering, this approach has several challenges:

- **Prioritization of acceptance criteria:** As noted above, agile holds that acceptance criteria are an important mechanism for determining a story's "conditions of satisfaction" determining when the story will be reviewed for acceptance by customers. However, there is no specific, well-accepted mechanism for differentially prioritizing those acceptance criteria.
- **Splitting a story vs. adding acceptance criteria:** As noted in the previous section, determining whether a large user story should be elaborated with acceptance criteria or split into additional, individual user stories is subject to vague criteria and, in part, driven to accommodate the length of a team's development iteration cycle (Cohn, 2017), Coupled with the prior point, this could make prioritizing an agile backlog difficult—we can prioritize easily if we split the story but not so easily if we capture the items as acceptance criteria.
- **Prioritizing large numbers of stories:** In large products, which may have hundreds of user stories, rank ordering them can be time consuming, tedious, and not very meaningful (Patton, 2008, p. 316). For example, realistically, there may be little practical difference in the value of, say, the two user stories competing for rank 112 in the product backlog. This may be exacerbated when stories are split too often (rather than annotated with acceptance criteria).

For their part, use cases have not stood still on this issue. As noted above, Jacobson and his collaborators have advocated for use case "slices," which define specific end-to-end paths through a single use case narrative (corresponding to a single user/feature/value user story). These involve defining slices by invoking one or more extensions. Each slice may be defined as its own backlog item with its own priority.

In this light, we can identify a path forward for improving user stories by leveraging use case slice concepts. Specifically, if a functional requirement represents a variation or exception to a single user/feature/value combination, then:

- That functional requirement should be recorded as an acceptance criterion attached to a base user story (this echoing the conclusion at the end of Section 3.3).
- The overall user story and its acceptance criteria can be mapped to a use case narrative, with the overall baseline user story mapping to the main success scenario and each functional acceptance criterion mapping to a use case extension.
- Each user story and its individual acceptance criteria (and the corresponding elements in the corresponding use case) should be prioritized independently.
- For large projects, we suggest utilizing MoSCoW prioritization (Clegg & Barker, 1994), rather than rank ordering, given the challenges of rank ordering large product backlogs.
- Logically, the base user story/main success scenario is the most important. Therefore, it should be of equal or greater importance than any of its extensions. Put more concretely, it doesn't make sense, for example, to have a base user story as a "Should Have" item, but with one or more extensions as "Must Have" items.

Overall, these points help address a lack of guidance from the agile world regarding how to split stories, which can be imprecise to the point for being unhelpful to practitioners and unteachable to students. For example, Bill Wake (2005) described no less than 20 different ways to split user stories, and in the years since the situation has not become any clearer, with various later trade book authors offering their own, unique splitting approaches.

For example, Leffingwell notes that, for user stories, "There is no set routine for splitting user stories into iteration-size bites…" (Leffingwell, 2011, p.111), and he suggests ten different possible ways of splitting user stories but offers no specific guidance regarding which one to use in any given situation. Similarly, Larman and Vodde (2010, pp. 247-65) offer sixteen approaches to segmenting work. However, as with Leffingwell, they do not provide guidance as to which approach to use in any given situation, nor do they show how such splitting can be accomplished using user stories.

In contrast, with use cases, the focus within a single use case on identifying base functional behavior vs. variations and exceptions to that base behavior leads to a consistent splitting strategy (slicing in Jacobson's terminology) of a use case based on functional variations and exceptions to the basic flow (main success scenario). By appealing to these use case principles, we can provide guidance "up-stream" to the writing

and structuring of user stories. This provides additional value in the complementary employment of user stories and use cases beyond the base idea of elaborating user stories via use cases.

3.5 Challenges of Acceptance Criteria: Formats and Non-functional

The discussion in the previous section provides important guidance on the use of functional acceptance criteria. However, there is more to clarify regarding acceptance criteria.

In agile, the definition of acceptance criteria (also called "conditions of satisfaction") is imprecise—even Mike Cohn indicates that he finds them confusing (Cohn, 2017). What are they? Most early writings speak not of acceptance criteria but of acceptance tests (Beck, 1999; Jeffries, 2001; Cohn, 2004). Today, acceptance criteria are understood to be "conditions of satisfaction" that give rise to a typically much larger number of detailed acceptance tests (Rubin, 2013, p. 85). But in terms of their content, it essentially appears that they can be anything: functional requirements, non-functional requirements, or "constraints." Further, with the exception of one school of thought, Behavioral Driven Development (BDD), described below, there also appears to be little to no guidance on the formatting of acceptance criteria.

This lack of precision makes teaching students and advising practitioners on acceptance criteria challenging. As with story splitting techniques, there simply is too little guidance and too much latitude. To address this, we again appeal to use case principles to help provide guidance and structure to this user story concept.

Specifically, we suggest that, given that use cases are functional requirements, we should delineate user acceptance criteria into two categories: functional and non-functional. As discussed in the previous section, for functional acceptance criteria, we should write and map acceptance criteria in the context of use cases as variations or exceptions/error handling to the main user story/use case.

However, we have not discussed the handling of non-functional requirements (NFRs)—which would include "FURPS" requirements such as usability, reliability, performance, and supportability. We note that use case sources distinguish between NFRs that apply to a single use case vs those that apply to the entire environment (see Bittner & Spence, p. 11; Cockburn, 1997, p. 13). If a requirement is non-functional but applies at the user story level, then per use case sources, we suggest including it in a non-functional requirement within a use case section dedicated for that purpose. In our webstore example above in Section 3.4.2, the final item "Process 100 orders per hour" is an NFR that would likely apply to that particular story.

If, however, a non-functional requirement applies to the entire project or environment, then we suggest determining whether it is a risk, issue, assumption, or constraint and then recording it in project-level documents. For example, a requirement for "99.9% uptime during normal business hours" would likely be a project-wide constraint.

We note that Daniel North in 2006 introduced the "given/when/then" format for functional acceptance tests (North, 2006) in the context of the advent of Behavior Driven Development (BDD), which was a derivative of Test Driven Development (TDD). Here, "Given" = context and preconditions, "When" = the behavior/event that triggers the scenario, "Then" = the expected outcome(s). While this was initially more focused on writing specific acceptance tests (rather than acceptance criteria forming the basis for acceptance tests), over time there appears to be some movement in quality assurance circles toward adopting this format for writing acceptance criteria, which could be similar to how the user/feature/value user story "formula" became a de facto standard.

For example, the acceptance criterion "Change shipping address alternative address" in the example above could be re-written "Given that the customer has recorded an alternative shipping address, when the customer picks that address, then the order should be updated to that address."

However, none of the textbooks or trade books we reviewed mention BDD or the given/when/then formulation, so we do not prescribe that specific approach. Rather, we focus on writing acceptance criteria as simple declarative statements, as per the example above, elaborated in use cases as a series of interaction steps. We do, however, note that, when determining whether an acceptance criterion is a functional vs. nonfunctional one, if it can be written in the given/when/then format, then it is a functional acceptance criterion.

3.6 Intelligent Modeling

The discussion in the sections above explicate key principles that set the stage for the prescriptive approach that we present below for utilizing user stories and use cases in a complementary manner. However, just because we know how to use user stories and use cases together, does that mean that we should always do so? More pointedly, are we arguing that a software team managing scope via a product backlog of user stories should then always create corresponding use cases? In short, a two-part answer to these questions would be: 1) "No, not always—elaborate with use cases in an intelligent manner," but also 2) "Consistent use of the principles may help create a stronger, better formed backlog of user stories."

We tackle each of these parts in turn. With respect to the first part, note that elaborating user stories with use case narratives will not be needed (or needed to the same degree) for all software features, even within in a single project. For example, for simple features or those that tap into familiar software design patterns, there may be little need for significant elaboration of a user story. Consider the familiar example of a registered website user needing to perform a password reset: "As user, I want to be able to request a password reset via an emailed link so that I can access the website." It is likely that this simple, familiar story might need little or no elaboration by a use case, even when the general project characteristics deviating from agile assumptions cited above are true. This appeals to the approach of modern use-case oriented methodologies such as the Unified Process, which may include significant detailed requirements documentation (including use cases), but which also advocate for focusing effort on major software project risks (Kroll & Kruchten, 2003, pp. 102-103).

Key use case authors support this view: Only elaborate user stories to use cases when:

1) Project characteristics dictate

and

2) Complexity and, therefore, risk of a specific use case dictates (Bittner & Spence, 2004, pp. 14-15; Cockburn pp. 16-17; Jacobson, Spence, & Bittner, 2011, p. 48).

As noted above, in this respect, the overall approach of *selectively* elaborating user stories with use cases marks a shift from the traditional requirements approaches. In them, systems analysts often create detailed requirements and designs for all software features prior to the initiation of software development and testing, based on the idea that all requirements are equally valuable and therefore are part of fixed scope (Boehm & Turner, 2004, 154).

Therefore, in our prescriptive approach in the next section, we want to qualify that we are advocating for an intelligent modeling approach to the employment of use case narratives to elaborate user stories—only creating formal, detailed requirements when they clearly add value.

On the other hand, even if we don't elaborate some or all user stories with use cases, thinking about user stories using use case principles can help clarify how best to identify, structure (aggregate or split), and add acceptance criteria a product backlog of user stories:

- **Identifying and structuring:** Can be done
  - **Top-down:** By identifying key user goals at a high level, for which we may be able to identify compound epics and then decompose user stories. This could be non-sequential (e.g., individual reports in a suite), but frequently is sequential when the epic represents a multi-step business process. Techniques for decomposition include Patton's (2008) user story mapping technique or activity diagrams and traditional flowcharts. First, we look for opportunities to split epics into lower-level user stories for a specific business goal with different actors—in this case, the lower-level user stories, themselves, may be subject to decomposition utilizing user story mapping or activity diagrams. For each such actor-goal combination, we may then split it into a series of lower-level user stories, each of which is distinguished by unique feature within that actor (put another way, each of which has a unique user/feature/value combination).
  - **Bottom-up:** Alternatively, when low-level user stories have been identified initially, we can aggregate them into sequential order in compound epics, which helps to verify that all key user stories have been identified and also helps to identify overall flow, variations, exceptions, and customer priorities.
- **Acceptance criteria:** Can be thought of and structured as follows

- **Functional:** The process above will result in a series of low-level user stories within epics. Below this level, any functional variations or exceptions to the user story is part of the same user/feature/reason combination. Thus, we would not split a user story at this level to smaller user stories but would instead annotate the user story with an acceptance criterion for each functional variation or exception.
- **Nonfunctional:** Any requirements item not directly impacting the behavior of the steps needed to execute user story would be clarified as a nonfunctional requirement (NFR). If that NFR applied to that story in particular, then if could be retained at that level. However, an NFR that applies to the entire project could be captured, typically as a constraint or, possibly, as a risk, issue, or assumption.

3.7 A Prescriptive Approach to Synergistic Use of User Stories and Use Cases

The prior section explained how use case principles can aid in the writing of user stories, even when corresponding use cases are not created. We now leverage that discussion as we tackle the final goal of section 3, in which we present a prescriptive approach for employing user stories and use cases in a synergistic manner.

This prescriptive approach leverages the principles articulated above and forms the basis for the case tutorial that we present in Section 4. Note that, as articulated below, the initial steps in the process describe a top-down approach. However, as noted in Section 3.7 above, these early steps can also be a bottom-up exercise. The important point is to understand compound epics in terms of business goals and actors so that lower-level, single user/feature/value combination user stories can be systematically identified in the context of the overall business process environment.

With these points in mind, here is our recommended prescriptive approach, expressed as abstract steps dissociated from any specific project context. Section 4 will provide a tutorial of this approach via a specific case-based example. Figure 6 provides a graphical illustration of the approach.

1) **Identify key business process activities as likely epics:** The top of Figure 6 shows an activity diagram that identifies key business process activities and their interrelationships. The use of swim lanes with multiple primary actors enables determining that this involves multiple users working across multiple sessions to accomplish the overall goal. Thus, in use case terms, we would need a summary level use case to represent this process. For each of the activities in this, we can identify user goal level use cases as potential compound epic user stories. As shown by the arrow from Goal 2 in the Summary Goal to User Goal 2 on the lower left, we are able to decompose this into multiple steps, confirming that it is a compound epic. Note that if we determine that these steps can be accomplished in a single “sitting,” then this would also correspond to a user goal level use case. However, if this requires multiple user sittings to accomplish, then we would likely split the epic for each sitting, with a corresponding set of split user goal level use cases.
2) **Identify individual user stories within each user goal level activity:** The diagram on the lower left shows an activity diagram or flowchart that identifies specific steps within the sitting. This depicts user-system interactions and, as a whole, will map to the user goal level use case. Items in the system swim lane represent system features or capabilities. Note that we designate each feature as either:
    - **Existing:** Meaning that it exists in the current state of the system and does not need to be included in the current project’s scope to create an improved future state environment.
    - **New:** Meaning that it is a new capability that may need to be included in scope, or
    - **Revised:** Meaning that it is an existing capability that needs to be enhanced or refactored. Therefore, these may also be included in project scope.
3) **Create user stories from each feature:** Each new or revised feature is created as a base user story. While this does not have to be in the user/feature/reason format, we recommend teaching this to students because of the prescriptive structure it provides.
4) **Identify and classify acceptance criteria for each user story:** Within each user story, identify acceptance criteria forming user conditions of satisfaction. For each acceptance criterion, classify it into one of three categories:

- **Functional:** Functional variations and exceptions to the base use case become extensions in the use case. Optionally, whether a criterion is functional or nonfunctional can be tested by determining if it can be expressed in the given/when/then format (functional acceptance criteria should be expressible in this format in the context of the functionality; nonfunctional requirements such as “Provide security for facing the public internet” will typically not fit easily into that format).
- **Non-functional goal level:** If a non-functional acceptance criterion applies specifically to this user goal, then it should be added in a special section in the use case outside of the main success scenario and extensions.
- **Non-functional project level:** However, if a nonfunctional acceptance criterion is identified as applying project- or environment-wide, then it should be added to project-level documentation, typically as a constraint.

5) **Create a user goal level use case narrative for each goal:** Each individual user story maps to a user goal level use case as follows:
   - **Base user stories:** Each base user story becomes a step in the main success scenario.
   - **Functional acceptance criteria:** Each functional acceptance criterion becomes an extension to its base user story step in the use case narrative. Additional details regarding the functional behavior of these may be included in the use case.
   - **Nonfunctional acceptance criteria:** As noted in the prior step, non-functional goal level acceptance criterion is included in a non-functional requirements section of the use case.
   - If any individual story is highly complex in terms of its acceptance criteria, consider splitting it out as an individual task-level story (see next point).
6) **Create individual subfunction use cases for highly complex stories/steps** from the goal-level use case. This should be done only if an individual user story is so complex that including its details in the regular user goal level would make that use case difficult to read. We do not show this in Figure 6, but, if needed, the subfunction use case would follow the same format as other use cases, and the goal-level “parent” use case would reference it at that step.
7) **Prioritize main success scenarios and extensions:** Each main success scenario step and corresponding extension may be prioritized. This is key to prioritizing individual acceptance criteria from the user stories. We suggest not ranking ordering for the reasons explained above. Rather, as discussed above, we recommend utilizing MoSCoW prioritization. Ensure that each main success scenario step has a priority greater than or equal to its extensions.

Although not an essential part of creating the user stories and use cases utilizing this approach, we note two additional follow-on activities that can be implemented and supported by this approach:

- **Identify extensions or slices to prioritize, estimate, and map to development iterations:** Any main success scenario in a use case likely to be implemented should be estimated. Beyond those, the team can prioritize either specific extensions or combinations of extensions—use case slices—to estimate. These estimates, which are likely to be more granular and detailed than user stories utilized alone, can then be used as a basis for release planning, including allocating items to development iterations.
- **Testing:** Identify a test case and create corresponding detailed test scripts for the main success scenario and for each extension (at least those that are likely to be included in scope) (Cockburn, 2001, pp. 178-9). If the team chose to utilize the BDD given/when/then format for writing user story acceptance criteria, then that can be leveraged, along with details from the user case extensions.

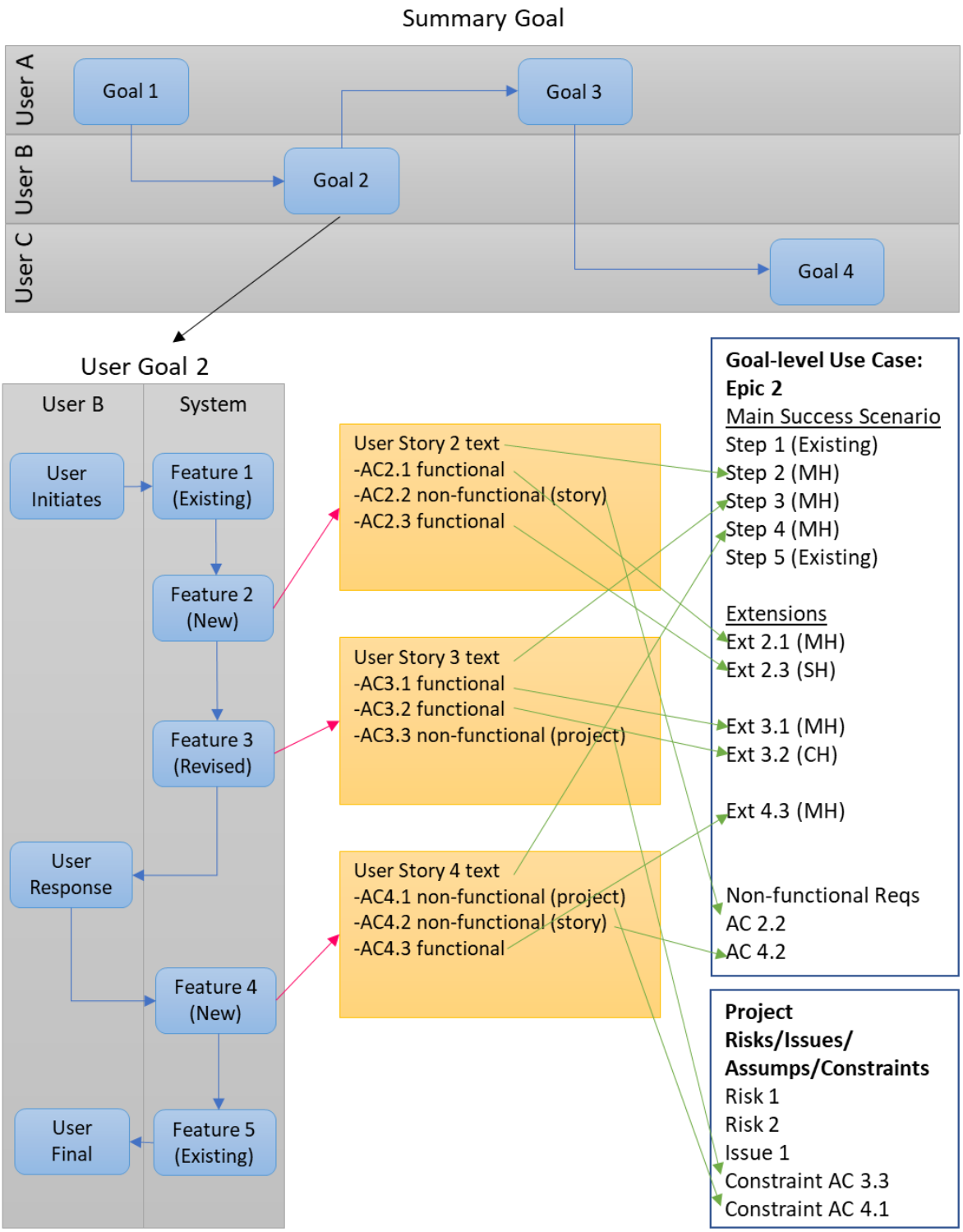

Figure 6. A Prescriptive Approach for Employing User Stories and Use Cases in a Synergistic Manner

3.8 Challenges and Drawbacks of Implementing the Proposed Methodology

Both user stories and use cases are highly generalizable techniques for describing functional requirements. Further, by explicitly linking the core components of each technique to the other, the synergistic approach described above and illustrated below is similarly intended to be highly generalizable. However, there are several specific challenges and drawbacks that practitioners should consider in implementing this proposed methodology:

- **Assessing costs vs. benefits:** First, implementing detailed use cases to elaborate user stories is a costly and time-consuming task. Per the discussion of Intelligent Modeling in Section 3.6, there needs to be a clear need in order to justify this effort. Typically, that would be established via the need to explicate specific epic user stories, either of the compound or complex variety. Of these two, the latter may provide better justification, because the team may be able to be handle the former through decomposition of the epic into several small and relatively simple user stories. More generally, Intelligent Modeling implies the ability of a team to recognize and act on the level of functional design processes and deliverables required in a specific situation. But implementing use cases in such a targeted manner based on cost/benefit analysis itself implies the need for additional effort to make those targeted determinations on a case-by-case basis.
- **Changing agile processes:** Implementing this approach may be challenging because of the software process change management required within a systems team. This may be especially acute within agile teams for at least three reasons:
    - **Learning use cases:** First, if the team has focused on agile techniques, then they may lack facility with use cases and will need to learn how to use them.
    - **Relearning user stories and linking to use cases:** Second, even if the team is skilled in use cases, they will have to revise their user story creation process to utilize use case principles, per Section 3.6, and then link those user stories to use cases, per Section 3.7.
    - **Changing team culture:** Third, and perhaps most importantly, many teams are deeply invested in a particular set of values and techniques—often holding these convictions in what Boehm and Turner (2004) characterize as “near-messianic stridency.” As such, many teams on the agile side of this debate may be culturally unprepared to embrace use cases, even in a targeted manner, again citing Leffingwell’s comment from Section 3.2 that use cases have been “largely banned from the agile tribes” (2011).
- **Project characteristics:** Finally, implementing this methodology may simply be inappropriate to the point of being counterproductive in projects whose characteristics fit what Boehm and Turner (2004) call the “agile home ground.” In particular, for projects with fundamentally unclear requirements upfront, the level of detail implied by use cases may simply be uncalled for. Further, even when requirements can be reasonably well understood up front, implementing use cases may not be worth it when those requirements are changing rapidly.

4 A Tutorial Case Example Illustrating the Prescriptive Approach

The approach in Section 3.7 provides an abstract explanation of our recommended prescriptive approach to utilizing user stories and use cases in a synergistic, complementary manner. Our main remaining task is to illustrate this approach in a specific, concrete manner utilizing a tutorial based on a case study.

4.1 Case Study: Enhancing a Graduate Application System

This case describes a project in which enhancements are needed for an existing graduate application system, or “GAS,” supporting student application processing and evaluation in a large graduate school environment. The key issue in the case is that the system currently provides little support for automating the process of evaluating student applications.

4.1.1 Actors

The case includes the following actor types:

- **System Administrator:** Responsible for maintaining the application system configuration tables, including those impacting the process of student application evaluations. This activity is a prerequisite for the other actors’ work.

- **Graduate School Staff:** Responsible for evaluating the *completeness* of each student application. For example, this could include checking for the presence/absence of required letters of recommendation, transcripts, standardized test scores, language test scores (for international applicants), and supporting documents such as resumes and statements of purpose. Once the graduate school staff member determines an application is complete, he/she forwards it to the relevant faculty.
- **Faculty Evaluators:** Responsible for evaluating the *quality* of each complete student application forwarded to them by the Graduate School. Put more concretely, determining whether the required grades, test scores, letters of recommendation, etc. meet the required standards for admission.

4.1.2 Summary Use Case and Candidate Epics

Note that after system setup by a System Administrator, the two evaluations occur in sequential order:

1. Graduate School Staff: Again, evaluating for application *completeness*.
2. Faculty Evaluator: Again, evaluating for application *quality*.

Conceptually, we can show this in Figure 7. Given that Figure 7 involves three different user roles, the implication is that we have at least three goal-level use cases, likely realized in requirements discussions first as epic user stories.

- As a System Administrator, I want to maintain rules in the system regarding rules for student application completeness and quality so that we can automate evaluations to the extent possible.
- As a Graduate School Staff member, I want the system to implement automated checks for the completeness of an application based on system rules so that I can save time.
- As a Faculty Evaluator, I want the system to implement automated checks on a completed application for the quality of an application so that I can more easily evaluate it.

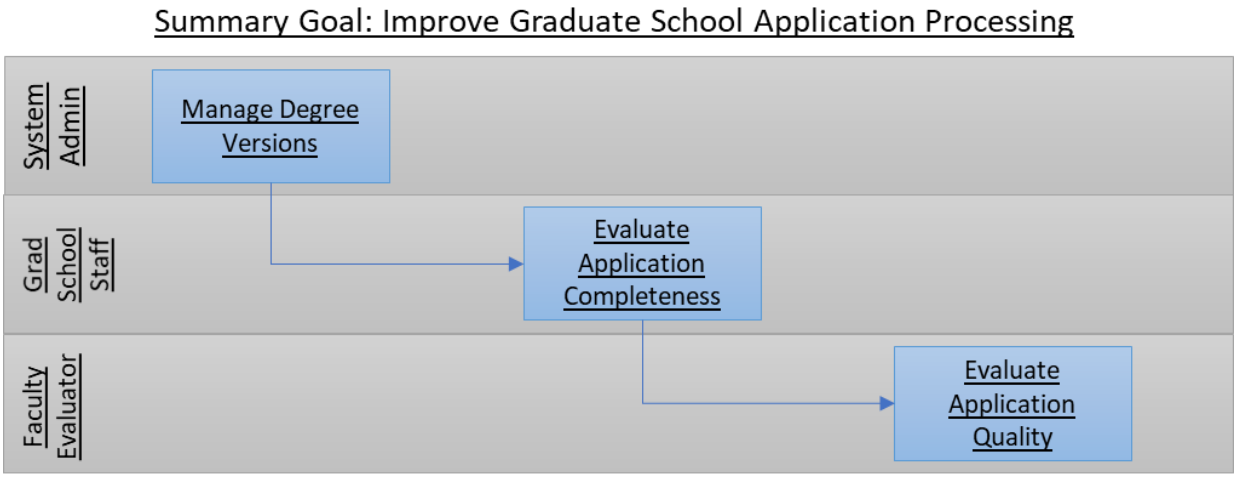

Figure 7. An Activity Diagram Illustrating Three Epics/Goal-level Use Cases

The fundamental problem blocking attainment of these features is that in its current state the system does not track the completeness and quality rules. Moreover, these cannot be implemented as additional attributes of the existing Degree entity, because the values for these settings vary along two dimensions:

- **Time:** For a given degree, the requirements for an application can vary over time, for example, the minimum GPA might currently be 3.0. It could, however, increase to a higher value in future terms (if the department offering that degree wished to increase the quality of students admitted).
- **Other variations at a point in time:** For example, for a given degree and a given term, the university might require three letters of recommendation for external applicants applying to a "traditional" version of that degree. But, if the university is offering an accelerated master's program (AMP) to its own undergraduate students from within the same major, it might choose to not require any letters of recommendation, given that the faculty evaluators would likely already be highly familiar with those students.

Because of this, the fundamental requirement is to implement support in the system for managing the rules for each version of the degree over time and at a point in time: The system administrator needs to be able to add and maintain these rules in the form of "degree versions," and the other two user stories need to implement automation utilizing these degree version rules.

4.1.3 Making Sense of Epics and User Stories

Table 3 presents a series of user stories derived from the case, with user stories grouped by epic. Note the following about these user stories:

- We present these stories as already identified, but as discussed earlier, the activity diagramming techniques we show could also be utilized to identify them in a top-down manner.
- Even given the summary discussion above and that there are relatively few stories—21 stories across three epics—it is difficult to make sense of these stories, even given the brevity of the "user/feature/value" use case formula. This recalls the problems pointed out by Jeff Patton of managing large numbers of "context-free" user stories.

The next two sections illustrate the use of activity diagrams, acceptance criteria, and use cases to help make sense of these requirements.

4.1.4 Activity Diagram and Use Case Mapping for the Faculty Evaluator Compound Epic

For space considerations, we only consider two of the three epics in the case: Manage Degree Versions and Evaluate Application Quality. We consider Evaluate Application Quality in this section and Manage Degree Versions in the next, based on how their characteristics illustrate "largeness" issues in terms of compound vs. complex epics, respectively.

The Evaluate Application Quality epic is portrayed in the activity diagram in Figure 8. This represents the same user stories as in the table, but, even though the diagram is fairly dense, we can make sense of what is going on much more easily than in the table of user stories because of the context the diagram supplies. The user initiates the process, in which, for each application, the system reads the degree version rules and evaluates items that can be easily automated: comparing test scores, GPAs, and the like against minimums. Consequently, the system provides those partially analyzed applications to the Faculty Evaluator, who is prompted by the system to review and record subjective evaluations for items such as specific courses taken previously, the resume, and so on. The process for an application ends with the system creating and sending a summary analysis to other faculty members on the committee.

The example highlights the utility of graphically decomposing a sequential process in this way. Again, we can either make sense of a list of user stories by casting them into an activity diagram, creating context, or we can utilize the activity diagram to decompose the process in the first place, thereby identifying the user stories.

However, there are limits to this approach: as we begin to capture too much detail, including processing variations and exceptions that would be included in acceptance criteria and use case extensions, the diagram becomes too dense, as illustrated in Figure 9, which attempts to include these details.

Instead, a better approach is to utilize user story acceptance criteria and map them to steps in the use case main success scenario and extensions. Because of space considerations, we focus on one particular step in this example: evaluating letters of recommendation ratings. These refer to subjective, 1-to-5 numerical ratings that the author of each recommendation letter is asked to submit regarding an applicant's capabilities in areas such as academics, leadership, communications, research, and so on.

Dimensions here include: the minimum number of letters of recommendation needed and the minimum mean subjective rating required.

In the most basic version of this, the applicant has submitted the minimum number of letters of recommendation and the mean of the ratings meets exceeds the minimum standard. Variations and exception conditions include:

- The number of letters of recommendation received fall below the minimum—as this should have been checked prior to the application being forwarded to the faculty, this represents an exception.
- The mean rating of those letters of recommendation falls below the minimum standard.

This could be recorded in a user story as shown in Figure 10.

Mapping these to a corresponding user goal use case for the Evaluate Application Quality epic, we could cast the user stories starting with the following **Main Success Scenario** steps (with MoSCoW categorizations in parentheses):

Table 3. Epics and User Stories for the GAS Case

EPIC/User Story	MoSCoW
MANAGEDEGVER: Manage Degree Version	
MANAGEDEGVER1 As a GAS Administrator, I need to be able to add or edit a degree version to enable the system to automatically evaluate grad school applications.	MH
MANAGEDEGVER2 As a GAS Administrator, I need to be able to create standardized and English proficiency tests as parameters for degree version test score requirements and scores submitted.	MH
MANAGEDEGVER3 As a GAS Administrator, I need to be able to record specific tests and minimum test scores for a given degree version so that we can evaluate an application.	MH
GSEVAL: Evaluate Application Completeness	
GSEVAL1 As a Grad School Evaluator, I need the system to automatically evaluate and update all my open applications on a daily basis to save me time and improve consistency.	SH
GSEVAL2 As a Grad School Evaluator, I need the system to determine if an English test is required for the current application's degree version and, if so, and whether it was supplied or not.	MH
GSEVAL3 As a Grad School Evaluator, I need the system to determine if a standardized test is required for the current application's degree version and if so, whether a required test score was supplied or not.	MH
GSEVAL4 As a Grad School Evaluator, I need the system to determine the number of letters of recommendation required (if any) for the current application's degree version and whether the required letters were submitted.	MH
GSEVAL5 As a Grad School Evaluator, I need the system determine if at least one transcript was submitted for the current application.	MH
GSEVAL6 As a Grad School Evaluator, I need the system to determine if a resume was required for the current application's degree version, and if so, whether it was supplied or not.	MH
GSEVAL7 As a Grad School Evaluator, I need the system to determine if a statement of purpose was required for the current application's degree version and whether it was supplied or not.	MH
GSEVAL8 As a Grad School Evaluator, I need the system to determine if all required items for this degree version were submitted and, if so, to update the application to Ready and email the faculty chair for this degree.	SH
GSEVAL9 As a Grad School Evaluator, if the degree version is designated as Quick Admit, then if the application meets all requirements, then I need the system to automatically admit the student and notify the faculty evaluators.	MH
FCEVAL: Evaluate Application Quality	
FCEVAL1 As a Faculty Evaluator, I need the system to automatically iterate through all Ready applications for a degree I select when I'm ready to evaluate.	SH
FCEVAL2 As a Faculty Evaluator, I need the system for each Ready application to compute letters of recommendation average rating and compare to the minimum for this degree version.	MH
FCEVAL3 As a Faculty Evaluator, I need the system for each Ready application where the applicant is not a native speaker to compare the test score to the minimum for this degree version.	MH
FCEVAL4 As a Faculty Evaluator, I need the system for each Ready application where the applicant to compare the standardized test score to the minimum for this degree version.	
FCEVAL5 As a Faculty Evaluator, I need the system for each Ready application compute the average overall GPA across transcripts and determine if it meets than the minimum for this degree version.	MH
FCEVAL6 As a Faculty Evaluator, for each transcript I need to be able to record in the system notes regarding specific UG courses and grades.	MH
FCEVAL7 As a Faculty Evaluator, I need to be able to record my subjective rating and notes in the system regarding the Statement of Purpose.	MH
FCEVAL8 As a Faculty Evaluator, for each transcript I need to be able to record my subjective rating in the system and notes in the system regarding the applicant's resume.	MH
FCEVAL9 As a Faculty Evaluator, I need to record a completed application evaluation.	MH
FCEVAL10 As a Faculty Evaluator, I need the system to generate and e-mail a summary analysis of each updated Ready application to the other committee members.	SH

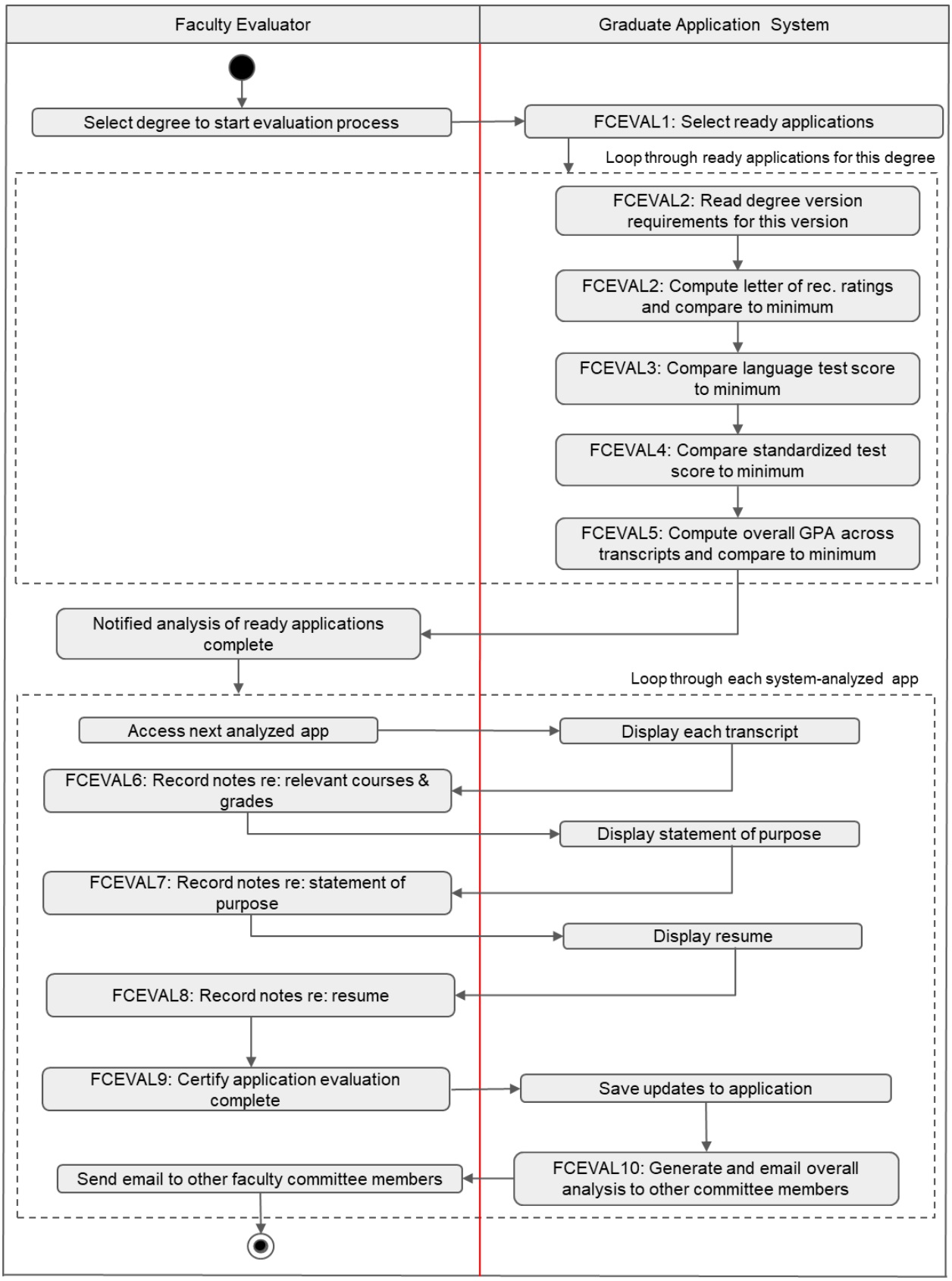

Figure 8. Activity Diagram for the Evaluate Application Quality Epic/User Goal

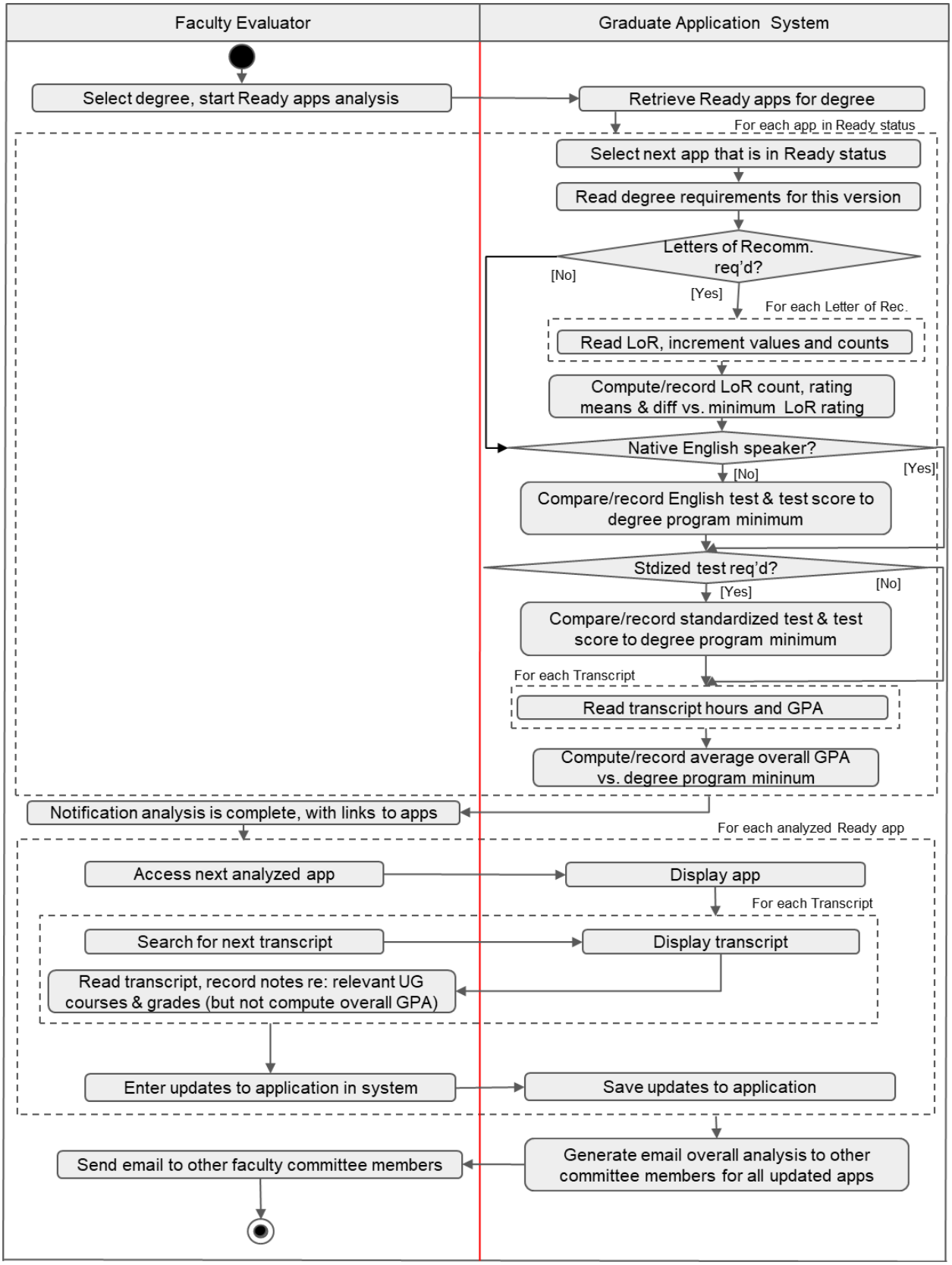

Figure 9. Activity Diagram for Evaluate Application Quality Epic/User Goal, Including Attempts to Portray Variations and Exceptions

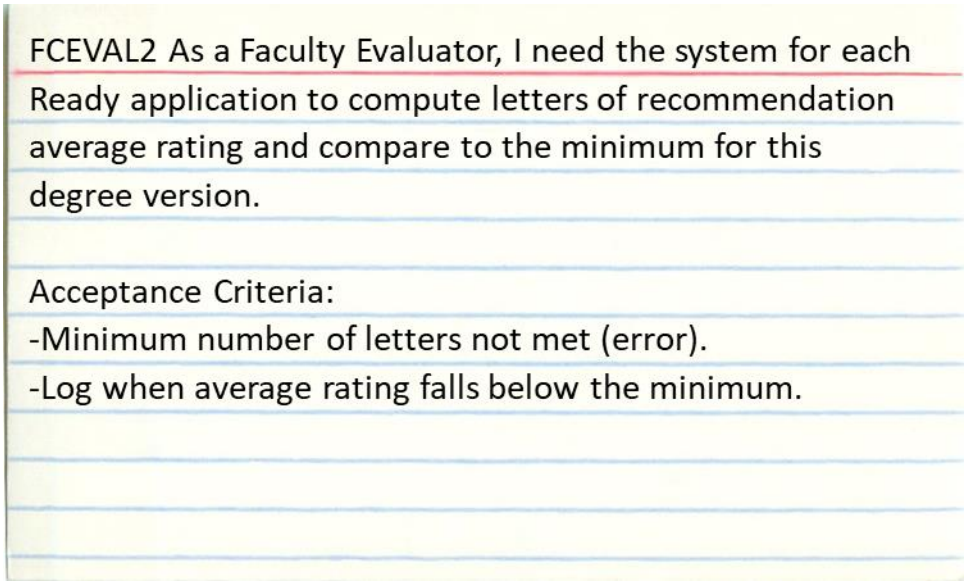

Figure 10. User Story for FCEVAL2 with Error and Variation Functional Acceptance Criteria

1. User selects degree to evaluate. (MH)
2. System selects all "ready" applications for this degree. The following steps apply to each application: (SH)
3. System accesses letters of recommendation for this application and computes the average of the subjective ratings for these letters. (MH)
4. (other user stories in the process would be shown as additional steps from here, with repeat from step 3 at the end of the loop).

For step 3, we could then add **extensions** as follows:

Error: Minimum number of letters of recommendation not received (CH)

3a. System compares a count of the letters of recommendation received to the minimum for this degree version and determines that the count of letters received is less than the minimum.

3a1. The system logs this as an error and skips to the next ready application.

Variation: Average ratings for letters of recommendation fall below the standard (SH)

3b. System compares the average ratings for the letters of recommendation to the minimum for this degree version and determines that the average rating falls below the standard.

3b1. The system logs the deficiency for use in the summary analysis.

In this example, note the following:

- The main success scenario step 3 may not be sufficient (without its extensions) to deploy the application to production. This is endorsed by Jacobson, Bittner, and Spence (2011), who support the approach of defining and constructing a main success scenario that is too simple to deploy to production, but then adding to it in subsequent sprints with slices containing extensions.
- We have assigned MoSCoW criteria to each step in the main success scenario and to each of the extensions. In the case of step 3, the main success scenario step is "must have," but then the others are of lower priority. For example, extension 3a as a "could have" should never occur, given that a previous process already validates that the needed number of letters of recommendation have been received.

This example demonstrates that the use case narrative provides the ability to elaborate specific user stories within an overall user goal, including to a level that cannot be readily achieved either through activity diagrams or flowcharts or within the user stories, themselves.

4.1.5 Elaborating a Complex Epic for Manage Degree Version

In the prior section, we showed how a goal-level use case can be used to elaborate individual user stories within a compound epic. In this example, we likely would not need to break out a separate, "subfunction" use case for any of the individual stories/use case steps. This is simply because they are not overly complex.

However, there are cases where the need to elaborate with use cases can arise not because of compound stories, but because a single story, including an individual story/use case step in a compound story, is highly complex.

We illustrate this with the Manage Degree Version epic. As a brief reminder, this represents a set of user stories to attach a set of application completeness/quality rules to a specific degree, like the MBA. We could, for example, require a GMAT score through a specific semester, but then drop that requirement for the next and ensuing semesters. We could also change rules regarding letters of recommendation, grade point average, and so on. The possibilities are practically infinite.

Now, adding a data entity as a child of the degree entity to store these rules is straightforward. But there is a particular area complexity that may motivate us to elaborate the user story MANAGEDDEGVAR1 "As a GAS Administrator, I need to be able to add or edit a degree version to enable the system to automatically evaluate grad school applications."

Specifically, sets of rules recorded in a degree version necessarily need to be tagged by a beginning date when the rules take effect and, optionally, an end date when the rules end (possibly because they are being superseded by a new set of rules).

When we add a degree version, it is natural to conceive of adding new rules (recorded as a new degree version) that supersedes an existing set of rules (an existing degree version). For example, we may have a degree version in place for an MSMIS degree with a begin date of 8/1/21 and an end date of 7/31/22. The simplest situation would be to add a new Degree Version record for the traditional MSMIS with a begin date of 8/1/22. But there are many possible complications here, such as (but not limited to):

- Adding the new degree version with a begin date in between the original dates (e.g., 1/1/21)
- Adding the new degree version when the old version didn't have an end date
- Adding the new degree version before an existing set of rules, either with or without an end date
- And so on—there are many circumstances to consider.

Our point is not to play out all of these complexities, but rather to simply raise them and then illustrate how a use case narrative could be used to elaborate these details. See Figure 11 for a use case narrative example that works through these rules.

In this example, the relevant section is the second extension, in which these rules are set forth. These are prime examples of extension rules that are not simple to convey by informal conversations—either from the business customer to a local team or especially when trying to convey such rules to remote IT team members. Even if the IT team member who discussed these rules with the customer is the same person developing the corresponding technical solution, it would likely behoove that team member to document these rules formally so they can save them for verification and use later.

5 Conclusion

This paper provides value to both industry practitioners and academics in two ways:

- Explains and clarifies the sometimes-confusing relationship between user stories and use cases.
- Leverages that explanation to propose and illustrate a detailed, prescriptive approach to synergistically employing user stories and use cases in a complementary manner. A key insight of this discussion is that employing use case principles in the creation of user stories can improve those user stories, even when those user stories are not then elaborated with use cases.

Use Case Narrative

Use case name: Add New Degree Version
Epic: MANAGE DEGREE VERSIONS
User Story: MANAGEDEGVER1 As a GAS administrator, I need to be able to add or edit a degree version to enable the system to automatically evaluate grad school applications.
Scope: Graduate Application System (GAS)
Primary actor: Graduate Application System Administrator
Preconditions: The Degree for which this Degree Version is being added exists in the system.
Minimal guarantee: The new Degree Version is not added and existing Degree Versions are unchanged.
Success guarantee: For a given Degree, a new Degree Version of a specific type is added with a Begin Date. The Begin Date and End Date (if added) of the new Degree Version does not conflict with the date range of any prior Degree Version associated with the same Degree.
Trigger: A GAS Administrator is notified by a department head to add a new Degree Version.

Main success scenario: Add a Degree Version (basic functionality and edits) MH

1. The GAS Admin accesses the Degrees screen, searches for the degree for which the Degree Version will be added, then selects edit for that degree.
2. GAS displays the Degree screen for that degree with a selection grid of existing Degree Versions.
3. The GAS Admin selects option to add a new Degree Version.
4. System displays a blank "new" Degree Version screen.
5. GAS Admin fills in editable field values.
6. GAS Admin clicks Save command button.
7. GAS saves the new record to the database.

Extensions:

Within a Degree Version Record, Implement Field Type and Edit Checks SH *(only one example shown to save space)*
7a. Begin Date of new Degree Version is blank.
7a1. GAS highlights error and displays error message.
7a2. Process continues from step 5 of the Main Success Scenario.

Across Degree Version Records, Implement Advanced Date Checking SH
7b. The new Degree Version Begin Date falls between a matching record's Begin Date and End Date.
7b1. GAS highlights error and displays error message.
7b2. Process continues from step 5 of the Main Success Scenario.
7c. The new Degree Version End Date is provided and falls between a matching record's Begin Date and End Date.
7c1. GAS highlights error and displays error message.
7c2. Process continues from step 5 of the Main Success Scenario.
7d. The new Degree Version lacks an End Date but there is a matching record with a later Begin Date.
7d1. GAS highlights error and displays error message.
7d2. Process continues from step 5 of the Main Success Scenario.
7e. The new Degree Version Begin Date and End Date surround the Begin Date and End Date of a matching record.
7e1. GAS highlights error and displays error message.
7e2. Process continues from step 5 of the Main Success Scenario.
7f. The new Degree Version record has a later Begin Date than a matching Degree Version record and that matching record lacks an End Date.
7f1. GAS updates the matching record's End Date to be the date prior to the new record's Begin Date.
7f2. GAS saves the new Degree Version record and the updated matching Degree Version record to the database.

Figure 11. Use Case Expanding on a Complex Epic

Particularly with respect to the second point, we note that prior literature—while generally advocating for utilizing user stories and use cases together—failed to provide detailed guidance for linking and integrating

these two requirements analysis techniques. By filling that gap in the literature, this paper makes a significant contribution to information systems research and practice.

We began by explaining that user stories and use cases are both key modern approaches to requirements analysis with which all SA&D students and practitioners need to be well-versed. Second, we explored the relationship between user stories and use cases, which can, but does not have to, be confusing—although their relationship has often been obscured by arguments between agile and plan-driven advocates, user stories originated in agile methods as brief use cases. Thus, when use cases are expressed in a brief format, they are virtually interchangeable with user stories. However, use cases can also be expressed in much greater detail than user stories, in which case the former can and, in certain circumstances, should be used to elaborate the latter. In particular, use cases can be useful in elaborating large, "epic" user stories, especially when the project is using a hybrid or plan-driven approach to systems development. Use cases are valuable both in dealing with compound user stories and with complex user stories.

Underlying this discussion is a firm belief that written specification of requirements independent of and preceding the development process will continue to be an important activity in many systems development projects—for many project types, agile emergent requirements and promises for discussion are not sufficient.

On the other hand, we acknowledge that use cases should not be utilized indiscriminately to elaborate user stories. Rather, creating detailed use cases is a costly and time-consuming task. Thus, we advocated for engaging in intelligent modeling, meaning that detailed use cases should be utilized only when creating them clearly adds value. Further, numerous projects in which requirements are fundamentally unclear or rapidly changing need to utilize a highly agile approach emphasizing user stories that are elaborated in an informal and emergent requirements manner. In these projects, use cases may not add value or may, in fact, actually negatively impact requirements and overall project success.

Further, we discussed how use case principles can be used in helping to create better sets of user stories, providing guidance in areas where the principles of defining user stories have been subject to imprecision. This is true even when use cases, themselves, are not utilized as a mechanism for discovering and structuring requirements. This guidance addresses questions such as determining how to split user stories, when to split a user story vs. adding acceptance criteria, prioritizing acceptance criteria, making sense of and prioritizing large numbers of user stories, and handling functional vs. non-functional requirements. Thus, even in projects where it is appropriate to utilize a highly agile, emergent requirements approach, employing use case principles can lead to better requirements.

Finally, we proposed at the conceptual level and illustrated with a practical example a simple, generalizable approach to utilizing user stories and use cases together in a complementary fashion. The key idea underlying this approach is a seamless, cohesive integration of business process modeling, user stories, and use case narratives into a set of requirements specifications that allows a software project to represent requirements at varying levels of specificity, depending on the needs of the overall project and each specific feature.

These use case principles and the complementary approach can be utilized to improve systems requirements both by industry practitioners and by information systems faculty in the teaching of systems analysis and design.

Acknowledgments

We acknowledge with gratitude the valuable feedback provided by the editorial review team at the *Communications of the Association for Information Systems*.

References

Beck, K. (1999) Embracing change with extreme programming. *IEEE Computer*. 32(10), 70-77.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M.,...,Thomas, D. (2001) The agile manifesto. *Agile Alliance*. Retrieved from http://agilemanifesto.org/

Bittner, K., & Spence, I. (2003). *Use case modeling*. Boston, MA: Pearson Education.

Boehm, R., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Boston, MA: Pearson Education.

Clegg, D., & Barker, R. (1994). *Case method fast-track: A RAD approach*. Boston, MA: Addison-Wesley.

Cockburn, A. (1997). *Surviving Object-Oriented Projects*. Boston, MA: Addison-Wesley.

Cockburn, A. (2001). *Writing effective use cases*. Boston, MA: Addison-Wesley.

Cohn, M. (2003) The upside of downsizing. *Software Test and Quality Engineering*. 5(1), 18-21.

Cohn, M. (2004). *User stories applied: For agile software development*. Boston, MA: Pearson Education.

Cohn, M. (2017) The two ways to add detail to user stories. *Mountain Goat Software*. Retrieved from https://www.mountaingoatsoftware.com/blog/the-two-ways-to-add-detail-to-user-stories#:~:text=And%20there%20are%20two%20ways,it%20or%20add%20acceptance%20criteria.

Dennis, A., Wixom, B. H., & Tegarden, D. (2015). *Systems analysis & design: An object-oriented approach with UML, 5th ed.* Hoboken, NJ: John Wiley & Sons.

Digital.ai (2020) 14th annual state of agile report. Published by Digital.ai. Retrieved from https://digital.ai/resource-center/analyst-reports/state-of-agile-report

Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language, 3rd ed*. Upper Saddle River, NJ: Pearson Education.

Jacobson, I. (1987) Object oriented development in an industrial environment. *OOPSLA '87 Proceedings*.

Jacobson, I., & Ng (2005) *Aspect-oriented software development with use cases*. Upper Saddle River, NJ: Addison-Wesley.

Jacobson, I., Christerson, M., Jonsson, P., & Övergaard, G. (1992) *Object-oriented software engineering: A use case driven approach*. New York: ACM Press.

Jacobson, I., Spence, I., & Bittner, K. (2011). Use-case 2.0: The guide to succeeding with use cases. *Ivar Jacobson International SA*. 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) Use-case 2.0. *Communications of the ACM*, 59(5), 61-69.

Jeffries, R. (2001) Essential XP: Card, conversation, confirmation. Ron Jeffries. Retrieved from https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/

Kendall, K. E., & Kendall, J. E. (2019). *Systems analysis and design, 10th ed*. Hoboken, NJ: Pearson Education.

Kroll, P., & Kruchten, P. (2003). *The Rational unified process made easy: A practitioner's guide to the RUP.* Upper Saddle River, NJ: Pearson Education.

Larman, C., & Vodde, B. (2010). *Practices for scaling lean & agile development: Large, multisite, and offshore product development with large-scale scrum*. Boston, MA: Pearson Education.

Leffingwell, D. (2007). *Scaling software agility: Best practices for large enterprises*. Boston, MA: Pearson Education.

Leffingwell, D. (2011). *Agile software requirements: Lean requirements practices for teams, programs, and the enterprise*. Boston, MA: Pearson Education.

Newkirk, James, and Robert C. Martin (2001) *Extreme Programming in Practice*. Upper Saddle River, N.J.: Addison-Wesley.

North, D. (2006) Introducing BDD. *Dan North & Associates*. Retrieved from https://dannorth.net/introducing-bdd/.

Patton, J. (2005) It's all in how you slice: Design your project in working layers to avoid half-baked incremental releases. *Better Software*. January 2005.

Patton, J. (2008) The new user story backlog is a map. *Jeff Patton & Associates*. Retrieved from https://www.jpattonassociates.com/the-new-backlog/.

Rubin, K. S. (2013). *Essential scrum: A practical guide to the most popular agile process*. Upper Saddle River, NJ: Pearson Education.

Satzinger, J. W., Jackson, R. B., & Burd, S. D. (2016). *Systems analysis and design in a changing world, 7th ed*. Boston, MA: Cengage Learning.

Schwaber, K. (1995) SCRUM development process. *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*.

Schwaber, K., & Beedle, M. (2002) *Agile software development with Scrum.* Upper Saddle River, NJ: Prentice Hall.

Spurrier, G., & Topi, H. (2017) When is agile appropriate for enterprise software development? *Proceedings of the 25th European Conference on Information Systems (ECIS).*

Spurrier, G., & Topi, H. (2018) Resolving the pedagogical disconnect between user stories and use cases in systems analysis and design textbooks. *Proceedings of the 24th Americas Conference on Information Systems (AMCIS)*.

Takeuchi, H. & Nonaka, I. (1986) The new new product development game. *Harvard Business Review,* 64(1).

Tilley, S., & Rosenblatt, H. (2017). *Systems analysis and design, 11th ed*. Boston, MA: Cedngage Learning.

Valacich, J. S., & George, J. F. (2017). *Modern systems analysis and design, 8th ed*. Boston, MA: Pearson Education.

Wake, B. (2003) INVEST in good stories, and SMART tasks. XP123. Retrieved from https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Wake, B. (2005) Twenty ways to split stories. XP123. Retrieved from https://xp123.com/articles/twenty-ways-to-split-stories/.

About the Authors

Dr. Gary Spurrier is an Assistant Professor of Practice of Management Information Systems in the Culverhouse College of Business at the University of Alabama. He earned his PhD in Management Information Systems at Indiana University in 1995. After earning his degree, he spent over 20 years in industry before returning to academia in 2017. His professional work experience includes roles as CIO, COO, product manager, software development project leader, and consultant. His research focuses on the practice and teaching of pragmatic and effective approaches to the development of large, complex application software systems. His research has been published in the *Journal of Information Systems Education*, the *Americas Conference on Information Systems*, and the *European Conference on Information Systems*. He is, with Dr. Topi, co-author of the textbook *Systems Analysis & Design in an Age of Options*.

Dr. Heikki Topi is Professor of Computer Information Systems at Bentley University. His Ph.D. in Management Information Systems is from Indiana University. His research focuses on systems development methodologies, information systems education, and human factors and usability in the context of enterprise systems. His research has been published in journals such as *European Journal of Information Systems*, *JASIST*, *Information Processing & Management*, *International Journal of Human-Computer Studies*, *Journal of Database Management*, and others. He is co-author of *Modern Database Management*, *Essentials of Database Management*, and *Systems Analysis & Design in an Age of Options* and co-editor of *IS Management Handbook* and *Computing Handbook*: *Information Systems and Information Technology*. He has been actively involved in global computing curriculum development and evaluation efforts since early 2000s (including *IS2002*, *CC2005 Overview Report*, *CC2020* and as task force co-chair of *IS2010* and *MSIS2016*). He serves currently on ABET's Computing Accreditation Commission and served earlier on ACM's Education Board and Council, on CSAB's Board of Directors, and on AIS Council as Vice President of Education.