Editor: **Sarah Gregory**
Intel Corporation
sarah.c.gregory@ieee.org

How Common Is Common Enough in Requirements-Engineering Practice?

Sarah Gregory

PART OF MY job at Intel is devoted to the curation of our requirements engineering (RE) training curriculum, including driving content development and mentoring new instructors. Although I currently work in our Internet of Things business, when I joined this group a year ago, I requested and received permission to continue to work across Intel. By delivering training company-wide and fielding requests for coaching from every part of our company, my business acumen remains strong, and I can keep an eye on the state of the RE practice across our many diverse groups. I can offer a bit more coaching where I see struggles, and transfer innovative practice from one group to others. Collaboration with my RE colleagues across the company helps us sustain "common enough" practice. It's harder than it sounds.

Definition Differences

When I teach, I often share the story of one of my first large-scale requirements efforts for an internal program. Dozens of stakeholders from across the company participated on the requirements elicitation team. The group gathered for a few days every other week to slog through the system definition, then in the following days worked separately. Requirements were captured in a somewhat consistent template, but in many different applications and tools—whatever the author or team knew best.

We were a few months into the effort before we realized, in the throes of a heated argument, that the battle was over three definitions of the word "characteristic." Each definition was completely correct for the context of its advocates. In two of the three cases, the definitions were part of standards for the respective disciplines. Once we discovered the confusion around "characteristic," we spotted several other instances in which one part of the company described what it did using language that other parts of the company used differently. We needed a decoder ring to talk among ourselves!

Compounding that problem was the presence of a relatively inexperienced requirements engineer (me) and a level of organizational and technical complexity I'd never faced before. The RE techniques I knew were sound, but I struggled to figure out just how I was supposed to be coaching this wildly diverse group of people. It was one program but with many constituencies and many varied practices. One group was a small team that developed software for a specific part of manufacturing operations. Another was a large corporate marketing team. Yet another represented a hardware product, and another came from Finance. At least 10 other groups or divisions were participating.

I had the template for a requirements artifact and eventually secured a tool in which I meticulously copied the email, Word, and Excel files I received. However, the template and tool weren't optimal for anyone. I just hoped they would be enough.

How Much Rigor?

In his seminal 2005 work, *Just Enough Requirements Management* (released after I survived that successful but difficult program), Al Davis described how factors such as corporate culture; product criticality, including the potential risks of missed requirements; and the implications of failing to meet users' needs all influence decisions about a company's RE practice.[1] Davis outlined a "range of rigor" along which a company's RE practices could land, providing just enough rigor to satisfactorily meet a company's objectives.

For companies developing safety-critical systems, strict standards often demand a very formal process across the range of RE activities, including rigorous inspections and techniques designed to mitigate risk. In contrast, companies that develop products with a higher tolerance for error and recovery might opt for lightweight development methods, including a much more lightweight approach to requirements. Reviews, documentation, and requirements themselves might be omitted, or at least significantly scaled back, in favor of rapid development cycles and assessing working products against user feedback. Over a decade later, Davis's work, especially his analysis of the range of rigor, remains a valuable resource for companies or project teams that need to determine the right level of process rigor and formality for their best requirements work. Even within sound agile practice, both the rigor and formality of practice might legitimately differ depending on the project and program needs.

Large complex enterprises that develop products across a broad spectrum of technology face a different challenge. Instead of determining just-enough requirements practice for the organization, the quest often becomes one to find a practice that is common enough to be applied across the enterprise. When safety-critical systems with standards-driven practices are one component of a company's portfolio, that company's decision might feel predetermined. Maintaining the tightest control over the requirements process through formal methods applied with precisely defined rigor might be demanded. However, for teams or divisions in the company that aren't subject to standards mandating the same formality, the additional overhead of process and tooling might not only feel oppressive but also be counterproductive, stifling innovation and responsiveness to immediate needs.

If the variety of products under development need to interoperate, the enterprise could legitimately have projects and teams that must land under "heavy process" while having projects and teams that cover the rest of the range back to, if not "no process," then perhaps "minimal process." That process might be contained entirely in lightweight methods such as Kanban or within the team's software development practices themselves (commenting code, test-driven development, and so on). In cases in which any particular team can operate essentially as a small independent company within the company, they might be free to follow Davis's guidelines and select methods and tools that provide just enough RE for their needs.

To the extent that the teams must collaborate, however, the challenge increases significantly. Rather than employing either no process or a heavy process, a company might need both. A proposed range for a common-enough process thus spans the distance between uniformity (a homogeneous environment with common tools, templates, and practices) and variance (a heterogeneous environment in which practices, templates, tools, and even terminology are chosen and used without attempts to align with others in use in the company).

Finding What's Common Enough

The challenge, then, becomes one of finding that elusive point of "common enough." We spent a number of years pondering "How common is common enough?" while working across the company with teams spanning the broadest range of our products and our internal systems. How much RE should we teach? What template—or templates—should we recommend? Where must we demand more rigor and precision, and when might teams be counseled to adopt lightweight RE practices?

The work of finding common-enough requirements practice must steer clear of two potential threats: dogma and corruption. Unlike with Davis's range of rigor or the spectrum of "common enough," dogma and corruption aren't necessarily polar opposites. Dogma intrudes when edicts about a given process or tool being "best" are "laid down by an authority as incontrovertibly true" (en.oxforddictionaries.com/definition/dogma). Questioning the dogma's validity can be perceived as threatening the overall system or enterprise itself.

The threat of corruption—deviation from an ideal with the potential of increased, unmanaged risks—is no less severe. "Corrupt" literally means "utterly broken." When any process or practice, regardless of its degree of rigor, is corrupted, the results are suspect. Perhaps unsurprisingly, when the threat of dogma is realized—practice adherence is demanded blindly, without regard for utility or the effects on the program or team—the risk of corruption—workarounds, skipped steps, and poor practices—increases too.

Finding a common-enough practice thus seems as likely as locating

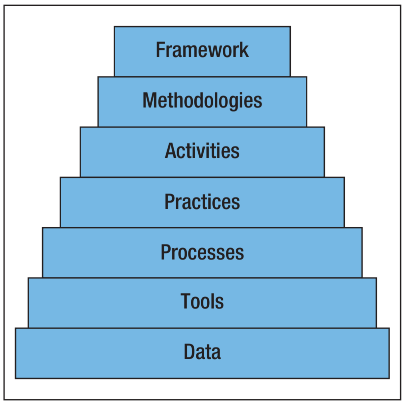

FIGURE 1. The layers of the product development process. This concept can help us focus on where and how to address commonality—or the lack thereof—of the requirements-engineering process and practices.

a unicorn tap-dancing through the aisles of a datacenter: wondrous to behold, but entirely unlikely.

Examining the Layers

What, then, is a large, heterogeneous environment to do? The beginnings of an answer to "How common is common enough?" can be found using the imperfect metaphor of a layer cake (see Figure 1) to disambiguate the overall product development process and focus on where and how to address commonality—or the lack thereof—of the RE process and practices. At each layer, consider the implications of too much commonality (dogma) or not enough (corruption).

We begin with a shared *framework*—typically, the highest level of a corporate lifecycle. Information at the framework level establishes the essential taxonomy and terminology used to communicate among team members and across group boundaries. Too much commonality appears as an attempt to force the framework to define all work and work products for all teams and engineers. Demanding absolute consistency in all areas might lose important domain or disciplinary nuance and involve unnecessary effort. Not enough commonality will produce silos in which each group adopts or invents a locally optimized solution, with little attempt to communicate across boundaries.

Beneath the framework we find *methodologies*; for the purpose of this analysis, we'll look at RE. Methodologies prescribe activities that enable the engineer or team to meet the intent of the framework's objectives. Too much commonality in methodologies occurs when they're prescribed as standards for all work that's to be done, regardless of the nature of the work, the team, or other characteristics of the project at hand. When the methodology contains insufficient commonality, every project begins anew, with no retrospective glance at the previous projects to see what might or might not have worked well and where opportunities to improve might exist.

Methodologies consist of *activities* that enable the engineer or team to fulfill the methodology's objectives by defining a range of practices in each activity. Several schemas break down RE into its constituent activities. The one I work with describes five parts: elicitation, analysis and validation, specification, verification, and management. Dogmatic insistence on eliciting all the requirements before beginning any design and development, similar adherence to perceived boundaries between the five parts, or treating those parts as a necessarily linear activity is an indication of too much commonality. No assessment of the activities at all to determine what teams will do to satisfy their objectives suggests insufficient commonality.

Practices are the actions we employ to fulfil the activities. They comprise the applied skills and techniques brought to bear on a particular project or program to meet a desired end. Prototyping, inspections (using the approaches of Tom Gilb and Dorothy Graham[2] or Michael Fagan[3]), ethnographic interviewing, semiformal notation, use case development, calculating story points, conducting a retrospective, and hundreds of other practices are available to use on any given project or program.

Each activity contains many practices, only some of which are suitable under a given set of circumstances. When someone asserts that because a particular practice worked before on a qualitatively and quantitatively different program, it will certainly work on the present one, you should suspect too much commonality. On the other hand, when each team member is free to determine how to do his or her work, inconsistent practices can lead to corruption of the work in the activity.

Successful completion of the RE practices is seen in tools and data—the repositories for, and output of, the practices, respectively. Questions of commonality often arise about tools. Must an organization settle on "one tool to rule them all"—a single-sourced system of record for product requirements data? Or, is a heterogeneous solution comprising many tools or tool suites possible? Similarly, to what extent can a data model be tailored to local or business group practices without that optimization hampering the whole organization's effectiveness?

In an enterprise with autonomy among the different organizations, finding a common-enough solution in any layer can be elusive. However, "enough" might not matter at many of the layers. A common framework allows everyone to understand milestones and deadlines at a program level. "Common enough" at the following layers of the cake invokes the many different memberships that are often at work in product development. These boundaries must be transcended to reach a shared understanding of the system under development. Boundaries between organizations and teams are obvious, but what about those between individuals, disciplines, phases, methodologies, conceptual models, datasets, activities, languages, the understanding of words, and so on? For example, a concept that's common in one team, location, or discipline might not be common in an adjacent team, location, or discipline.

The answer we're left with when considering "How common is common enough?" for RE practices in a complex enterprise is deceptively simple: It depends. The specific factors on which the answer will depend can vary but are often found at the points where the different boundaries can be crossed. More complex dynamics must be considered, including explorations of the sociotechnical aspects of the discipline of RE itself. Once "common enough" is determined, the overall results might well improve. After all, our goal is to deliver products, not requirements! A resolution takes the form of heuristics rather than rules. I'll address some of those heuristics in my next article.

ABOUT THE AUTHOR

SARAH GREGORY is a staff requirements engineer at Intel Corporation. Contact her at sarah.c.gregory@ieee.org.

Acknowledgments

This article is based on my presentation at the 20th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ).[4] I'm also deeply indebted to my colleague and former mentor, Erik Simmons (Construx, USA), for discussions about the question of "enough requirements" and his significant contribution to the analysis herein.

References

1. A.M. Davis, *Just Enough Requirements Management: Where Software Development Meets Marketing*, Dorset House, 2005.
2. T. Gilb and D. Graham, *Software Inspection*, Addison-Wesley, 1994.
3. M. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems J.*, vol. 15, no. 3, 1976, pp. 182–211.
4. S. Gregory, "How Common is Common Enough?," presentation at 20th Int'l Working Conf. Requirements Eng.: Foundation for Software Quality (REFSQ), 2014.