The Practitioner’s Cycles, Part 1: Actual World Problems

Steven V. Deal, Deal Corp.
Robert R. Hoffman, Institute for Human and Machine Cognition

This is one of two essays on the forces and constraints of procurement versus the goals of human centering, including the creation of intelligent technologies that are usable, useful, and understandable. The procurement process tends to de-emphasize these goals while focusing on strict adherence to rules and regulations. As a result, software system development processes, described in texts and acquisition documents, have come to be misaligned with the challenges faced by development teams. This misalignment between “actual world problems” and normative documentation repeatedly results in failed systems. A real-life practitioner’s account illustrates this point by describing how a group of individuals, acting on their own initiative and at their own risk, short-circuited the rules and constraints of the procurement process to turn a procurement process failure into a success.

In the next essay, we will present a model called the Practitioner’s Cycles and discuss how this model applies to the envisioned world problem, which is the challenge of creating intelligent technologies for new work systems.

Actual World Problems

System failures and the concomitant waste of competitive and taxpayer resources seem to be the fare of the day. Woes about issues in major procurements are dramatic due to the magnitude of squandering and the profound setbacks to operational capabilities. Information technology, especially intelligent systems technology, seems to be particularly susceptible to programmatic breakdowns.

Several surveys have documented information technology procurement issues. Although some reports criticize the methods that the surveys use, methodological weakness succumbs to the evidentiary tide.

Information technology procurement failure is gauged in various ways. For information technology administrators, operators, and others, deliverables are unsatisfying, related to gaps of usefulness, usability, and understandability. For management, projects fail to achieve their business objectives, and procurements exceed the costs budgeted for their execution. Moreover, projects take far longer than expected to complete.

Several fallacies must be overcome if the chances for project success are to increase. Such fallacies include the belief that “requirements creep” is a bad, unnecessary thing that should be avoided, when it is actually a necessity as well as an empirical fact. There is also a fallacy that one can conduct “concurrent maturation” of enabling technology without anchoring this in cognitive task analysis to understand the actual work of the domain. Project development teams often misunderstand the perspectives of the various stakeholders. Moreover, there is a fallacy that budgeting should be exclusively project centered without embracing human-centering goals. These actual world problems involve a mismatch between the challenges at hand and the available resources and delegated responsibilities.

The Challenge Is Always Today’s

Technologies must be relevant to immediate needs. Businesses must address market demand even if, as with innovative new products, they must create that demand themselves. Government sponsors want to solve the problems their people are dealing with today. Technologists have to align themselves with
rapid “satisficing” (settling on a solution they think is “good enough”) and deliver usable, useful, and understandable products that are achievable in reality.

The Response Is Delayed

Despite these immediate demands, however, it can take years to get a new system into the hands of workers. The 2008 US Government Accountability Office Assessment found that for 39 programs begun since 2001, average system development time was 37 months, a duration that is typical of large software projects. The acquisition process takes years, during which attempts are made to capture all stakeholder requirements, include all the latest technologies, and neutralize risks. New technologies emerge, and sponsors desiring to maintain and obtain additional competitive advantage will contractually insist that these be incorporated into existing product designs, extending the system development process even further. US Army Vice Chief of Staff General Peter Chiarelli stated the system development process even further. US Army Vice Chief of Staff General Peter Chiarelli stated the problem bluntly: “We have to find better ways to keep up with technology. It doesn’t do us any good to have a procurement cycle that takes 10 to 15 years.”

The Challenge Is Morphing

Moreover, during the years it takes to bring a product to market or to achieve an initial operating capability, the people who need new systems and upgrades are vulnerable to adaptive circumstances and adversaries. This is a reflection of the moving-target law that governs sociotechnical workplaces. Today’s needs must be repeatedly redefined to adapt to changing mission requirements and the obsolescence introduced because the process is so lengthy. The operational context changes, and projects must compete for resources. Expertise gets lost due to attrition. New competitors enter the market place; old competitors change their tactics. New alliances emerge. Management incentives reward new and additional tasks, leading to misaligned goals. Job responsibilities and incumbents change.

Complexity Will Kill You

In addition to the above actual world problems, things seem inevitably to become more complex. Congressman Curt Weldon was quoted by the Department of the Air Force as saying: “The [Joint Strike Fighter software task alone is five times larger than that required for] software requirements.” Thus, one of the risk factors affecting the ability to transition research is the difficulty of rapidly validating and verifying the functionality of technologies against realistic problems.

System Development Never Really Ends Anyway

Finally, system development never really ends anyway. The process lives on in rebuilds, procedural modifications, block upgrades, replacements, and altered missions or applications. This has been called the Law of Trickle-Off Ergonomics. This never-ending process sort of nullifies the very meaning of the term deliverable. One of the most difficult tasks a developer faces is to fully list the assumptions on which system development is based. The complexity of current and emerging systems, and systems of systems, requires breakthrough approaches in system development, including organizational structures, personnel development plans and career tracks, and budgeting mechanisms that span the entire product-line life cycle.

Is Acquisition Reform an Impossibility?

Quite a few US government programs illustrate the consequences of ignoring actual world problems: cost overruns, low usability, software issues, and so on. The roster includes cases in which expensive ongoing programs had to be drastically restructured. However, a few government initiatives also present sincere attempts to escape actual world problems.

For instance, recent acquisition programs of the US Department of Defense have sought to apply a method or goal of “rapid acquisition.” Indeed, all the DoD branches seek novel methods for rigorous yet rapid testing of technology (for instance, testing for interactions between different software systems). In fact, the DoD has a Rapid Reaction Technology Office charged with addressing this very problem. This push has been motivated by the desire to get new systems (including weapons systems but also robotic systems, software systems, decision aids, and others) to the warfighters as rapidly as possible. This push has also been motivated by recognition of the burdens and roadblocks involved in acquiring complex information technologies.

Alas, reform can end up being a mere piggyback on the status quo. This, of course, is the easy path. And it’s illustrated by calls for acquisition reform that effectively just entail additional regulations and reporting requirements, new bureaucracy for cost estimation, more oversight of testing, additional requirements for prototyping (with escape clauses), and more punishments for overruns—and, this, despite there already being a shortage in acquisition personnel. Then, killer complexity comes into
play, because the remaining people are unable to keep pace with the regulation sprawl.

New regulations often come in response to a breakdown or other disaster, but they generally try to solve the problem in piecemeal fashion and in ways that don’t take into account the actual underlying factors behind the disaster—which are typically systemic but get reduced to a blame game. Also left out of consideration is the issue of how existing regulations will interact with the new piece of regulation—for instance, by creating goal conflicts.

To illustrate this piggybacking approach, we can tap into the 1997 General Accounting Office report on opportunities to reform acquisition.15 This report recommended two actions: conduct joint mission assessments, and test sufficiently before production. How would these apparently simple and substantive reforms actually be implemented if they were to occur under the umbrella of the current procurement system?

First, “joint” would need to be defined. How many offices or branches would need to participate for an assessment to be considered “joint”? Once the stakeholders were identified, common understandings would have to be achieved. Good luck with that. Next, the steps constituting a mission assessment would need to be delimited and explicated. Then, the criteria for sufficiency would be investigated and detailed (that is, pried from context). Also, the stipulation “before production” would have to be specified: should it be before full production or before low-rate production? Furthermore, the means to measure compliance would have to be identified and carefully specified.

All of this would require timely documentation-processing mechanisms. At the high end, policy would need to be coordinated across all branches. Language accounting for the cultures and specialized procedures of each branch must be agreed upon. Each branch would draft procedures with legalistic precision, which must be reviewed, revised, and finalized. Defense acquisition coursework would be updated. Clauses for agreements and contracts would be revised. Acquisition professionals would need to be made aware of the new guidelines, and contractors would need to become familiar with the new policies.

The end result would be a recreation of the hobbled and intractable process that triggered all the grumbling in the first place: an explosion of documentation, additional regulations, and more oversight.

The root limitation of the piggybacking approach is that legislation, policy, standards, and processes are limited in their capacity to effect change that does not “complexify.” Various reforms have been tried and have failed, leading to advice by Harvey Sapolsky,16 professor of Public Policy and Organization Emeritus at Massachusetts Institute of Technology, to skip acquisition reform, on the basis of the following assertions:

- Everything has been tried. The truth is you can’t fix the acquisition system. All the insiders know this.
- Sponsors ask for crazy systems that must do impossible things.
- Existing systems are so good that new systems must be really, really good to justify the government buying new ones.
- Only a few firms have the capability to manage complex, new projects.
- The vendors’ claim, “We can build that, cheaper than you think.”
- Acquisition rules intentionally slow things down with demands for constant reconsideration, in the hope that support for the project will fade.

When we really want something quickly, like new systems urgently needed by warfighters, we have to suspend the rules, set up a fast track, and push aside the bureaucrats.

- The problem with fast tracking as a solution is that everyone must agree that a weapon is needed. Most of the time, there is disagreement.
- Skipping the reform charade might force officials to educate the taxpayers instead of hoodwink them, because making decisions on which weapons to develop and buy is very difficult: we don’t know what wars we’ll fight and what weapons we’ll have to counter.

This circumstance is not peculiar to government. David Riley, director of Pera International’s acquisitions support group, noted in speaking of manufacturing in the United Kingdom: “The current pattern is to spend 80 percent of management time and costs on financing taxation, legal and contractual issues, and 20 percent on the market, technology and people issues. Successful acquisition usually has the reverse equation.”17

Alternatively, reform can be radical: the unfixable system gets trashed, and the core rules of the game get changed. This requires acknowledging actual world problems, which include satisfying stakeholders by maintaining jobs; sustaining an industrial technology base; and acknowledging that excruciating oversight could waste even more dollars than abuse, capable systems take a long time to create, and the world doesn’t wait while you create them.

No wonder insiders believe fixing “the system” is impossible.

What Makes for Success?

On the basis of a survey of software development activities that were
arguably successful, Robert Frese and Vicki Sauter presented several success elements centering on communication, planning, and management support. Kelly Neville, Jennifer Fowlkes, and Robert Hoffman conducted in-depth interviews with experienced program managers and software engineers about procurement, and they too listed some success elements, including accommodation to changing requirements, management buy-in and support, and communication and coordination among team members. Some of the success elements revealed in these explorations are fine as general guidance, such as “solicit input to fully understand user needs;” “promote communication among executives, managers, developers, suppliers and users;” and “pay attention to human considerations.” But as Neville, Fowlkes, and Hoffman point out, methods and tools for actually accomplishing such communication and facilitation goals are entirely lacking. The result is that program managers and researchers must fend for themselves.

Companies such as Apple have demonstrated the value of addressing one particular element, user needs, with products such as the iPod and the iPhone. The importance of understanding user needs is highlighted by cases in which the users of the technology were the ones who were innovative, whether in business or in the military. Technologists and inventors are often surprised at the uses to which people put their innovations. Think back to the original purposes of GPS and the Internet, and compare them with today’s applications. People discover techniques and opportunities afforded by product features, which they gradually employ to their (often novel) advantage.

Failure to engage in rich cognitive task analysis to reveal “desires” can be practically guaranteed to result in software systems that are limited in usability, usefulness, and understandability. Success cases at coping with actual world problems tend to be those in which the technology developers had a deep understanding of the nature of the user’s work. The “Practitioner’s Tale” sidebar illustrates this point.

The practitioner’s tale given in the sidebar is an example of success at coping with actual world problems. Success was achieved by deviating from the rules while appreciating their intent. The technologists complained and said “No!” The program manager felt ill at ease, since he was put at considerable risk. But he and his team “got away with it.” By implication, perhaps the only way to achieve human centering in the face of actual world problems is by deviating from the rules, because of the strictures of the mandated or traditional concepts and processes of procurement.

The next essay in this department will continue this extrapolation all the way to a model we call the Practitioner’s Cycles, which represents one path for escaping actual world problems but also for coping with the envisioned world problem.

References
The Practitioner’s Tale

The following story was told to us by a government program manager with decades of experience at leading software development teams. This manager had first-hand experience (and frustration) with the actual world problems we discuss in this essay. In our retelling of the tale, we have scrubbed certain identifying information to protect the guilty. With the exception of this and some minor editorial changes and paraphrasing, we retell this real-life account in the practitioner’s own words.

The job at an aviation radar facility was to ensure safety by coordinating aviation operations. Though the analog scopes worked well, they were hard wired. It was difficult to integrate the data. The operators wanted more flexibility. So a contractor was tasked to create a totally new radar facility—the building, the system, the scopes, the operations floor, the works—with one large screen, one large computer to integrate the data.

So, at the Preliminary Design Review, they set the ground rules. Developers had to follow the government specifications. They had to use a waterfall development model. It was a nightmare. Everything had to be modular, so the people who built the new building never talked to the people who would use it—the people who were still using the old scopes at the old building. The old building, with the current operators, stood less than 100 yards away, but no one ever went over to find out how the operators performed their jobs.

The new facility was built and the users came in. They took one look at the new radar picture and knew it was wrong. For instance, the code was generating bogus tracks due to weather. The operator could not credibly keep the real tracks clear. The procurement people had followed the rules: do the specifications and build to the requirements by government specifications. The result was trash.

So they brought in my team to fix the code. We looked at the code and predicted its failure modes, which the operators subsequently confirmed but hadn’t spoken up about. Our team said the new building was OK, the big screen was OK, but everything else had to go, especially a lot of the software. But the parts of the code that had to be trashed could not be isolated. Though it had been modularized, it relied on a shared memory. Overwrites and shares meant that the functions could not be split out—like trying to remove an octopus from a barrel of them without disturbing any of the others. The developers had violated the intent of the rules because they followed the rules blindly.

The program manager asked what they would have to do to replace it all. Our team had anticipated this. We had to work with the actual operators to find the intent of the coded rules. Our coders needed to understand the needs of the radar operators. We sent some junior engineers to the site, telling them, “Go there, get training, and come back when you have reached the point where the director would say he would let you do the actual job.”

That took about three weeks. With their input, our team reached agreement on what the system really needed to do, using the operator’s jargon. The engineers came back two weeks later with a summary document that made sense. Then they went back to the operators again and got a list of their “desirements” that could be incorporated in the rebuild. We played a set of older mission data into the new tracker. Our team was able to craft the look and feel that the operators wanted. We checked the buttonology with the operators and had the first build in three months.

The program manager said we could not write code prior to the Preliminary Design Review. That would be a violation of the government-specified waterfall. The intent of that is to not write code before the problem is specified. But our development team already knew more about the problem than the procurement and contracting people. The project directors set these ground rules; we were going to do it their way. But when we got to the Critical Design Review (when you decide on the hardware), our team was told we had to decide on the machines and the operating system. We said “No! The technology is improving too fast.” We wanted to test the alternative hardware later on. In a few months there would be better computers available and a compiler extension that would simplify the code and make compilation easier in the long run.

The program director was aghast. But our team continued in another refinement of the build. One desirement was for a training capability, the reuse of tough cases. We did some of that, at no extra cost. We tested a rebuild, delivered it, and it went operational. Some operators came over from the old analog facility. One said he could not do his job with the new screen. But we were prepared. Our system had tools to develop screens, because we had anticipated the need. We had built in the flexibility. So this operator came in the next day and we had made the screen he wanted. He was shocked, and loved it. “How did you do that so fast?” That’s what our screen development tools allowed.

A radar facility at another location had also started all over to recreate a new system. But they were not learning from the users. They fell back on the rules and the comfortable process. Some operators that had worked with our team moved to this other location and called in our team to observe and help. (Our team had made the software so that it could be exported to other facilities.)

Then there was some new leadership on the government side. People started moving to our new spiral model, which by then our team had been using for some years. Users cannot specify requirements, as these are understood by software people. Users can specify their needs and their desirements. Now, people were starting to do this, but we saw a tendency for them to warp this new approach back into their traditional procedures. You need to go by the intent of the rules, not the rules themselves.


Steven V. Deal is vice president and systems engineer at Deal Corp. Contact him at s.deal@sbcglobal.net.

Robert R. Hoffman is a senior research scientist at the Institute for Human and Machine Cognition. Contact him at rhoffman@ihmc.us.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.