A Manifesto "about knowledge/expertise things"

Organizations prosper because of their ability to mine, structure, refine or create knowledge and put it to use faster than the competition. But too often the subject matter experts who most contribute to that knowledge are trapped by cumbersome processes and must work with poor tools.

Worse, they are relegated to supplying their knowledge to software developers who then analyse, interpret and understand it before implementing it in the system that ultimately makes money for the organization.

This setup is terribly inefficient. It causes errors, requires days, weeks or months for changes that should take hours, and frustrates everybody involved. As the rate of change increases, time-to-market is expected to shorten and product variability blooms, it is clear that this is not how things should work.

We advocate for change. Here is how we achieve it:

Provide quality **tools** for subject matter experts to capture knowledge, to help them understand and reason about it, and to find inconsistencies in their knowledge by supporting iterative capture, convenient editing, live execution and testing,.	**as opposed to** expecting experts to juggle complex subject matter in their minds, Word documents, user stories and other non-tool-processable formats.
Remove friction by **automating** the downstream system development and deployment processes in order to ensure that subject matter experts are in control of "their" part of the software,	**as opposed to** forcing them to wait for developers get around to understanding the application logic and then implementing, testing and deploying the code.
Let the software developers **build the tools**, robust execution platforms and the automation between the two,	**as opposed to** having them involved in every instance of application logic development over and over again.

Elaborations

Is this contradicting Agile? Over the last decade, agile approaches that focus on interdisciplinary, end-to-end-responsible and empowered teams have become mainstream in many organizations. And sometimes a focus on tools, automation and formal/executable descriptions is understood as contradicting agile practices and values, as driving the business and engineers apart. In the context of this approach tools are key enablers. They help us achieve better understanding within the whole organization, through the usage of a common language, and permit significantly faster iterations. If anything it makes possible to reach the goals promoted by agile. However, this is a false dichotomy. Precise and analysable models of the subject matter are the perfect basis for understanding and productive collaboration between subject matter experts and software developers. And nothing in this approach prohibits incremental development, short release cycles and integration of feedback. To the contrary:~~,~~ the approach makes changes less painful, cheaper, faster, and safer.

Will this drive developers and subject matter experts apart? Our proposed approach lets everybody focus on their strong suit: subject matter experts will spend less time trivializing their knowledge, explaining it to developers and spending energy on pushing things through the process. They will have more time exercising, growing and operationalizing the knowledge, creating value for the organization. In our experience, many subject matter experts really want to do this, they are willing to be precise and eager to take responsibility. They just aren't given the tools. On the other hand, the approach lets software engineers focus on building robust platforms and automation pipelines instead of understanding and then implementing every particular change in business logic. So we don't drive the two communities apart -~~,~~ to the contrary. The common understanding captured in the tool language acts as a contract between the communities,~~;~~ clarifying responsibilities, and providing to everyone the framework to do~~the~~ what they do best. Ultimately, our approach leads to increased professional satisfaction and pride for both subject matter and software engineering experts.

What about process? As a corollary, tools and automation alone are not the full solution either. Because they change what and how much a single person can achieve, it can affect the relationship and communication paths between roles and departments. So the organizational structures and responsibilities as well as the required processes must support this new paradigm. Which is why the adoption of this approach must be coordinated with business process refactorings. The opposite is also true: business process improvement can be made much more potent by introducing the kinds of tools we emphasize in this manifesto.

This is a major cultural change! In many organizations, putting the subject matter experts at the center and expecting them to deliver correct and potentially executable models is a big change. They have to learn new tools and are now responsible for the correct behavior of the product. Some SMEs might not believe this is possible or might be afraid of the learning curve. Software developers might fear that they are "automated away" or might be scared of the challenge of becoming toolsmiths. Some~~one~~ will resist this change, because of their own personal interests. They may end up leaving, and this is ok. To avoid these factors from stopping the initiative, it must be managed carefully in terms of education, training, coaching and people's sensitivities.

Are we turning subject matter experts into programmers? Is this approach a sneaky way of turning subject matter experts into software developers? Not at all. Expressing knowledge in a way that is~~n~~ precise, analyz~~s~~able and correct is not the same as software engineering. In fact, there are lots of domains in which computers and software tools are the central tool in the subject matter experts' toolbox already. Obvious examples include CAD programs for mechanical engineers, Matlab for control engineers, and R for statisticians. We propose a similar approach for knowledge workers in domains. We've seen it in healthcare, finance, tax, public administration and game design. But more and more domains are increasingly computational, including biology, medicine and law. So instead of actually making them programmers by requiring them to use Python or C++, we propose custom-designed tools for knowledge workers in various domains. It's not fair to expect them to keep track of (continuously evolving) subject matter with whiteboards, Word and Excel documents or DOORS databases. Believe us when we say that we've seen completely intractable monsters in all of these tools, even in safety-critical domains!

Doesn't machine learning solve the problem? There are two kinds of knowledge. One kind of knowledge is deductive, it is derived from (often large amounts of) existing data. The notion of right or wrong is statistical. There are no hard and fast rules. To use such knowledge meaningfully, it is enough to be approximately right with high-enough probability, and you determine quality through testing only. You don't have to be able to give reasons for a decision. Machine learning is great for such cases. The other kind of knowledge is based on explicit and deterministic rules (even though those might hide in hard-to-untangle verbal descriptions). There's an objective measure of right and wrong, it is necessary to be right 100% of the time and you have to be able to explain a decision. In addition to testing, you can also analyze the rule set and reason about it. This is the kind of knowledge we address with our proposal here.

Can't we just use off-the-shelf tools? Can we use one of the low-code environments out there? No. Upon closer inspection, it turns out that the vast majority of low code platforms do not address the specific subject matter of an oganizations, but instead enable non-professionals to build ad-hoc solutions to ancillary IT and data management problems. Most of these tools are essentially Microsoft Access in the cloud. Business rule engines are a step in the right direction, because they also try to empower business anali~~y~~sts. But just like business process modeling tools, they are generic. In contrast, we advocate toolifying the core of your domain, the part that makes you successful as a business and that distinguishes you from your competitors. By definition you need custom tools, not generic tools for managing data input/output through web forms.

Objections to tool building. An objection we hear often relates to the development of tools: "We build XYZ, we're not a tool building company". This is rather shortsighted. Many industries develop their own tools, or are at least deeply involved in co-developing them with a supplier who specializes in tools. And often these tools are custom-developed to the particular product the tools are supposed to help build. Mechanical engineering companies have departments full of people who plan and develop manufacturing lines. And there are two more important conclusions we can draw from the analogy. First, just as in the manufacturing industry, tools are usually not built from scratch either. They rely on tool platforms and tools for building tools. There are many such tool-building-tools in software, many under the headlines of modeling, language engineering, business rules engines or knowledge engineering. Second, the investment in custom tools only pays off if there is some notion of economy of scale, i.e., if they can be used for building lots of products. However, this is very much true for many organizations: they build hundreds of treatment algorithm apps, tax calculation software for dozens of different taxes over years and decades, or hundreds of different but related insurance products.

Why should software engineers care? One of the most important foundations of software engineering is separation of concerns. Roughly speaking, it means that you shouldn't put stuff that doesn't belong together in the same place. Don't mix apples and oranges. In our opinion the most important concerns to separate is subject matter and "software stuff", because you can't untangle the two later; the problem has been described as unscrambling the scrambled egg in order to get egg white and yolk back. Keeping the sources of the two separate (and mixing them only as a transient artifact) is crucial because it allows for the empowerment of subject matter experts as described in this manifesto at length. But it also means that you can change and evolve the technical platform without affecting the subject matter itself -- the two lifecycles are largely decoupled, as they should be. Considering the technology cycles today, this is absolutely crucial for keeping your software stack up to date. In the end, the approach really avoids running into the legacy trap, which we'd describe as having sunk all your knowledge in a semantically un-analysable and now outdated mountains of source code.