I Believe

I THOROUGHLY ENJOYED Dr. Shull’s editorial “I Believe” in the January/February 2012 issue of IEEE Software. It is something I’ve been struggling with throughout my career, both in industry and especially now that I am in academia.

Belief is a powerful word that tends to lead to unquestionable conclusions that are regarded as facts. A belief is more like an axiom than a theorem, and as an axiom, it is considered certain. Beliefs, in the traditional sense, aren’t supposed to be verified; they aren’t even supposed to be questioned. In fact, the very act of questioning a belief, more often than not, turns into an almost religious argument—even when the belief is about software development. Perhaps the reason people often feel attacked and need to defend rather than discuss or debate is precisely because they think of their arguments as beliefs.

Intuition, on the other hand, is built from a wealth of experience and serves as a readily accessible repository of knowledge, a repository that we use to reach decisions quickly. The beauty of intuition is that it is approximate and treated as such by definition. Most people also agree that intuition can only be gained with experience and that although it’s not necessarily rational, it is instinctual.

We often question and verify our intuition when the task calls for a more rationalized result. Perhaps we could rely more on our intuition and less on our beliefs. Our intuition allows us to “develop a sense of healthy skepticism” and to “trust but verify.” Our beliefs: not so much. Maybe we should begin discussions with “I think” rather than “I believe.” This small difference could lead to discussions that feel more like healthy debates and less like “holy wars.”

The question then becomes whether practices such as test-driven development (TDD) and excessive coupling are beliefs or intuitions and whether they’re reusable as such. I think it is the role of science to take beliefs and intuition and turn them into theories and then facts that engineers can use to make choices. Because our science is so young, we have very few facts, but we do have well-defined principles that can and should be reused.

Principles depend on context. They’ll work in one and not work (as well) in another. For example, the principles of classical mechanics are different from those of quantum mechanics, and yet they’re both valid. The difference is in the context. It would be interesting to study what principles in TDD could or could not be reused, depending on context.

Furthermore, in my opinion, there’s a difference between “excessive coupling leading to poorly maintainable code” and TDD. The former is a principle, while the latter is an application of many principles.

The problem with trying to reuse something like TDD is that it tries to reuse a set of principles wholesale. TDD is a solution that aggregates a set of principles. Trying to reuse them as a whole is like trying to reuse the implementation of one project for another; it can’t be done wholesale, only parts of it will be reusable.

That’s why in practice, most teams would agree that they’re using a mix of processes. If we analyze the reusability and therefore applicability of individual principles across different domains, we might be able to formally define contexts in software engineering and link those contexts with well-established sets of fully reusable principles.

Magdin Stoica
Sheridan Institute of Technology
and Advanced Learning