Position Paper for SEMAT, Zurich, 2010.03.17-18

Meilir Page-Jones

As I read through the various correspondence and writings concerning the SEMAT initiative for our first gathering (Zurich, 2010.03.17-18), I observed that we have not yet reached complete unanimity on which particular problem(s) the SEMAT initiative intends to solve.

My assumption in this brief paper is that we're attempting to correct some or all of the following:

1. Historically, the practitioners of software development have resisted the introduction of discipline into their practices.
2. Where the introduction has occurred at all, it has been at a superficial level. Perhaps this is partly due to the above problem. Or perhaps it is due to a lack (quantity or quality) of training. Or perhaps it is due to the caliber of many of the people in the industry.
3. Due in part to the above two problems, there has been a "thrashing of methods" in the industry, whereby a new method will make unsubstantiated claims *for* itself and *against* a current method in order to supplant the latter. Under examination, the two methods may be found to be doing essentially the same thing, with only surface differences. The industry tends to gratuitously cast out existing methods for no reason other than "they are old". Sometimes even, a rich but older method is put aside in favor of a new method with less actual content. Thus, our profession has taken on the characteristics not of science, of engineering or even of craftsmanship but of art and fashion. Knowledge and skill have been lost, not gained, over recent decades.

My understanding is that we are <u>not</u> attempting directly:

- To improve the quality of software or its production. (Nevertheless, we trust that this will be the ultimate consequence of our initiative.)
- To define any particular approach to creating or managing software.

So, given the above situation and goals, my position is that we will attempt to:

- Taxonomize the core concepts that any contemporary software method is likely to contain. (Examples would be: coupling, cohesion, encapsulation, domain separation, unitization of deliverables and so forth.)
- Taxonomize the core procedures that the conduct of any contemporary software method is likely to demand. (Examples would be: elaboration of unconstrained requirements in a given domain, mapping of those requirements to the constraints of a lower domain, the sequencing strategy for allocating project effort and so forth.)
- Identify the engineering circumstances in which a given concept or procedure lends positive or negative advantage, thereby offering criteria for assessing how suitable a given method in given circumstances.
- Perhaps "deconstruct" two or three popular approaches to software (both older and contemporary ones) in order to assess how useful our taxonomy really is.

Implicit in this is the idea of a *core software-engineering curriculum*, which is a set of disciplines in which practitioners should be trained in order to impart to them the understanding needed to categorize and assess any software method - past, present or future.

Meilir Page-Jones, 2010.01.15