The Business of Software

The next generation of software engineering will involve designing systems without using paper-based formats, instead using software to develop software.

It is somewhat ironic that in our business we don’t use much software to produce software. Excepting dynamic testing, we don’t generally employ executable knowledge to create executable knowledge. We have compilers that convert a text form of instructions understandable to (some) humans into machine instructions executable by a machine. But a compiler doesn’t know what we are supposed to do and it cannot generate or validate the external real-world functional knowledge. We have CASE tools and modeling tools that allow us to store some of the knowledge we need in a variety of formats such as directed graphs, tables, and dictionaries. But we have to put the knowledge into these tools—the tools cannot self-populate. And often, the only thing we can do with knowledge we have put into these tools is merely retrieve it and look at it.

Knowledge in most software development tools does not usually execute. A tool may verify that the knowledge format conforms to some conventional modeling rules, but few tools allow us to automatically validate that the contained knowledge is correct and applicable to the situation at hand in the real world.

Electronic Paper

As rich and clever (and expensive) as these tools may be, they mostly fail the test of executability. They are more like electronic paper than they are software. The real work of software development is done in word processors, list-format data catalogs, and simple constrained drawing tools. More correctly, most of the work in software development is done in the individual and collective brains of the developers—the word processors and other tools are the non-executable book-like places we put the knowledge once we have it.

An IDE Whose Time Has Not Quite Come

Integrated development environments (IDEs) have come a long way since the days of programming using the VI editor, but they are still mostly word processors with some specific look-up and retrieval processes and just a few executing functions such as condition checking (which, of course, we have to define).

We spend most of our systems development effort collecting pieces of knowledge and putting them down on pieces of paper (electronic or otherwise) and it is interesting to consider just what the physical act of putting knowledge onto pieces of paper does to the knowledge (see the figure here).

Words are in sequence. When we put information into words on pieces of paper we apply a sequence. Words are in sequence. We can change the sequence words are in but in sequence words are nonetheless. So even if what we are describing is not sequential, when we put it in words, it will be.

Text is control oriented. The act of reading (and writing) words is a control-oriented task. There is always an associated point of attention—what is being read at that moment in time. The point of control generally moves from one statement to another according to the sequential behavior described previously. Putting ideas onto paper forces this control focus on the ideas we put on the paper.

Reading is single tasking. People do not usually read just one word at a time; they actually absorb patterns or blocks of several words at the same time. But our attention is mostly fixed.
The systems we are building now are those for which the ontology of paper does not work very well and we have to rethink how we think about systems.