Hot Topic: theorY Driven Development (YDD)

In the twenty-five years since Brooks published his "No Silver Bullet" paper, there has been no systematic effort to address what he believed to be THE essential difficulty of software – the conceptual construct.

Some have floated ideas – most notably Peter Naur's "programming as theory building" – and others have suggested practices that (notably in Agile) could be useful in resolving THE essential difficulty. Until now, however, there has been little or no effort to integrate and articulate an approach that would help individuals and teams to establish a "conceptual construct" or "theory" and drive design, implementation, and testing of that theory/construct.

At Chili we will explore and discuss this issue with the objective of creating a comprehensive definition of YDD. The expected deliverable from our efforts will be a heavily annotated outline for a book with all participants being co-authors.

I will circulate a set of background readings to participants at least two weeks before we meet in Carefree. Until then, some pertinent quotes from Brook's and Naur to stimulate your thinking.

Brooks:
The essence of a software entity is a construct of interlocking concepts ... I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation.

"As soon as we attempt to diagram software structure, we find it to constitute not one, but several, general directed graphs superimposed one upon another. ... The graphs are usually not even planar, much less hierarchical ..."

The complexity of software is an essential property, not an accidental one. Hence, descriptions of a software entity that abstract away its complexity often abstract away its essence. For three centuries, mathematics and the physical sciences made great strides by constructing simplified models of complex phenomena, deriving properties from the models, and verifying those properties by experiment. This paradigm worked because the complexities ignored in the models were not the essential properties of the phenomena. It does not work when the complexities are the essence.

Much of the complexity that [the software developer] must master is arbitrary complexity, forced without rhyme or reason by the many human institutions and systems to which his interfaces must conform. These differ from interface to interface, and from time to time, not because of necessity but only because they were designed by different people, rather than by God.

Naur:

"I shall use the word programming to denote the whole activity of design and implementation of programmed solutions. ... the activity of matching some significant part and aspect of an activity in the real world to the formal symbol manipulation that can be done by a program running on a computer.

... If it is granted that programming must involve, as the essential part, a building up of the programmer's knowledge, the next issue is to characterize that knowledge --- the

programmer's knowledge should be regarded as a theory, ,,, Very briefly, a person who has or possesses a theory in this sense knows how to do certain things and in addition can support the actual doing with explanations, justifications, and answers to queries about the activity of concern.

1) The programmer having the theory of the program can explain how the solution relates to the affairs of the world that it helps to handle. ...
2) The programmer having the theory of the program can explain ***why*** *each part of the program is what it is ...*
3) The programmer having the theory of the program is able to response constructively to any demand for a modification of the program so as to support the affairs of the world in a new manner.

Therefore, the most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements. ... it is necessary to allow for an extensive iteration between the client and the designer as part of the system definition.

Incremental development – grow, don't build, software."