David Lambertson

**Software Design: An Introduction Summary**

**Important Points from Article**

* “This end goal is one that we need to keep in mind in seeking to provide a concise review of some of the many factors and issues that are involved in designing software-based systems.” [209]
* “An important task for a designer is to formulate and develop some form of abstract design model that represents his or her ideas about a solution.” [209]
* “These techniques include the use of abstract ‘mental models’ of their solutions, which can then be mentally executed to simulate the final system behavior, reusing parts of previous solutions; and making notes about future (detailed) intentions as reminders for later stages in development.” [210]
* “One important point that should be made here: Designing software is rarely a completely unconstrained process.” [210]
* “Constraints can affect the design process as well as the form of the product. Designing software is not an isolated and independent activity.” [210]
* “The activities of analysis may be used to identify the form of solution that will meet the user’s needs, and the designer is then required to provide a solution that conforms to that form.” [210]
* “In a like manner, a designer must provide a set of specifications for those who are to construct a system. These specifications for those who are to construct a system. These needs to be as clear, complete, and unambiguous as possible, but of course it is likely that further needs for change will be identified during implementation.” [210]
* “In this section we examine some of the ways in which a designer’s ideas about design model can be visualized by using various forms of description.” [211]
* “The three principal forms of description normally used to realize the design viewpoints are text, diagrams, and mathematical expressions.” [211]
* “There is a long tradition of drawing diagrams to provide abstractions in science and engineering, and even through the ‘invisibility’ factor makes the form of these less intuitive when used to describe software, they are still very useful.” [212]
* “Mathematical notations are of course ideally suited to providing concise abstractions of ideas, and so it is hardly surprising that these have been employed in what we generally term the Formal Description Techniques, or FDTs.” [213]
* “This notation captures one aspect of constructional information, namely the invocation hierarchy that exists between subprogram units.” [214]
* “The partial analogy with programming points to one of the problems that hinders teaching about design, namely that of scale.” [216]
* “Each step alters the design method, either by elaborating the details of the model, or by transforming them, adding new attributes to create new viewpoint descriptions.” [217]
* “A characteristic of second-generation design methods is that they involve constructing much more complex design models from the start, usually involving the use of more than one design viewpoint.” [218]
* “It can be argued that object-oriented analysis and design techniques are still evolving (perhaps not as rapidly as was once hoped).” [219]

**Things I Didn't Agree With**

“We do not have ‘testing methods’ or even ‘programming methods’. When teaching programming, we commonly provide the students with a set of ‘programming metaphors’.” [216]

I do not agree with this statement by the author because testing and programming do have methods. We learn that there is Waterfall, code-and-fix, and many others. While we call these software models, they also fall under the method category.

**Things I Did Not Understand**

I understood the entire article.