David Lambertson

**No Silver Bullet – Essence and Accident in Software Engineering Summary**

**Important Points from Article**

* “The familiar software project has something of this character (at least as seen by the nontechnical manager), usually innocent and straightforward, but capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet, something to make software costs drop as rapidly as computer hardware costs do.” [180-181]
* “Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any-no inventions that will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware.” [181]
* “Software entities are more complex for their size than perhaps any other human construct, because no two parts are alike (at least above the statement level).” [182]
* “Many of the classical problems of developing software products derive from the essential complexity and its nonlinear increases with size.” [183]
* “No such faith comforts the software engineer. Much of the complexity he must master is arbitrary complexity, forced without rhyme or reason by the many human institutions and systems to which his interfaces must conform.” [184]
* “In short, the software product is embedded in a cultural matrix of applications, users, laws, and machine vehicles. These all change continually, and their changes inexorable force change upon the software product.” [185]
* “Surely, the most powerful stroke for software productivity, reliability, and simplicity has been progressive use of high-level languages for programming.” [186]
* “Time-sharing preserves immediacy, and hence enables us to maintain an overview of complexity.” [187]
* “Ada indeed not only reflects evolutionary improvements in language concepts but embodies features to encourage modern design and modularization concepts.” [188]
* “The concept of the abstract data type is that an object’s type should be defined by a name, a set of proper values, and a set of proper operations, rather than its storage structure, which should be hidden.” [189]
* “Many people expect advances in artificial intelligence to provide the revolutionary breakthrough and quality. I do not.” [190]
* An expert system is a program containing a generalized inference engine and a rule base, designed to take input data and assumptions and explore the logical consequences through the inferences derivable from the rule base, yielding conclusions and advice, and offering to explain its results by retracing its reasoning for the user.” [191]
* “The work required to generate the diagnostic rules is work that will have to be done anyway in generating the set of test cases for the modules and for the system.” [192]
* “The most powerful contribution of expert systems will surely be to put at the service of the inexperienced programmer the experience and accumulated wisdom of the best programmers.” [193]
* “Sometimes the promise of such an approach is postulated from the analogy with VLSI chip design, where computer graphics plays so fruitful a role.” [194]
* “Program verification is a very powerful concept, and it will be very important for such things as secure operating system kernels.” [195]
* “Compiling could stand a boost, but a factor of 10 in machine speed would surely leave think-time the dominant activity in the programmer’s day. Indeed, it appears to be so now.” [196]
* “Every day this becomes easier, as more and more vendors offer more and better software products for a dizzying variety of applications.” [197]
* “Computers are now so commonplace, if not yet so beloved, that the adaptations are accepted as a matter of course.” [198]
* “The hardest single part of building a software system is deciding precisely what to build.” [199]
* “The purpose of the prototype is to make real the conceptual structure specified, so that the client can test it for consistency and usability.” [200]
* “That is, the system should first be made to run, even though it does nothing useful except call the proper set of dummy subprograms.” [201]
* “We can get good designs by following good practices instead of poor ones. Good design practices can be taught.” [202]
* “Most organizations spend considerable effort in finding and cultivating the management prospects; I know of none that spends equal effort in finding and developing the great designers upon whom the technical excellence of the products will ultimately depend.” [203]

**Things I Didn't Agree With**

“Moreover, at some point the elaboration of a high-level language becomes a burden that increases, not reduces, the intellectual task of the user who rarely uses the esoteric constructs.” [186-187]

I do not agree with this statement because I believe that high-level languages increase intellectual task. By using a high-level language, it is much easier to describe what we are doing in terms others will understand. By developing for understanding also, we are able to make sure when someone else has to use our code, it is readable and not complex.

**Things I Did Not Understand**

I understood the whole article.