Brady Field

Software Engineering

**Important points**

“Software project management encompasses the following activities: measurement, project estimating, risk analysis, scheduling, tracking, and control.” [15]

“The role of estimating within the software process often serves as a ‘sanity check’ on the predefined deadlines and budgets that have been established by management.” [16]

“Computing technology and program architectures have undergone a sea change, but the characteristics that define high-quality software appear to be invariant.” [17]

“Many software developers use software testing as a quality assurance ‘safety net.’” [18]

“Every change to software has the potential for introducing error or creating side effects that propagate errors.” [18]

“A problem cannot be fully defined and bounded until it is communicated.” [19]

“In essence, software engineers and the customer must develop an effective mechanism for defining and negotiating the basic requirements for the software project.” [19]

“There are many different ways to look at the requirements for a computer-based system.” [20]

“Different modes of representation force the software team to consider requirements from different viewpoints, an approach that has a higher probability of uncovering omissions, inconsistencies, and ambiguity.” [20]

“Software design is a set of basic principles and a pyramid of modeling methods that provide the necessary framework for ‘getting it right.’” [21]

“The design of tests for software and other engineered products can be as challenging as the initial design of the product itself. Recalling the objectives of testing, we must design tests that have the highest likelihood of finding the most errors with a minimal about of time and effort.” [23]

“Stated in another way, a design pattern describes a design structure that solves a particular design problem within a specific context and amid ‘forces’ that may have an impact on the manner in which the pattern is applied and used.” [24]

“If technology problems associated with reuse are overcome (and this is likely), management and cultural challenges remain. Who will have responsibility for creating reusable components? Who will manage them once they are created? Who will bear the additional costs of developing reusable components? What incentives will be provided for software engineers to use them? How will revenues be generated from reuse? What are the risks associated with creating a reuse culture? How will developers of reusable component be compensated? How will legal issues such as liability and copyright protection be addressed? These and many other questions remain to be answered.” [25]

“Tools for the next ten years will address all aspects of the methods landscape but they should emphasize reuse and reengineering.” [26]

**Disagreements**

“The glory years of third-generation programming languages are rapidly coming to a close. Fourth generation techniques, graphical methods, component-based software construction, and a variety of other approaches have already captured a significant percentage of all software construction activities, and there is little debate that their penetration will grow.” [22]

Ok, when I read this I actually started laughing. Some examples of third-generation programming languages are C, C++, C#, and Java. These are all still very widely used today for many general-purpose applications. While there are many newer languages out there, these languages are good examples of third-generation programming languages that are not going anywhere anytime soon.

**Questions**

I have no questions. I understood everything in the article.