Brady Field

Alternative Software Life Cycle Models

**Important points**

“Software development, for any application, is an expensive and risky endeavor.” [289]

“The classic waterfall model for the software life cycle (see Figure 2.3.1) was defined as early as 1970 by Dr. Winston Royce [1] to help cope with the growing complexity of the aerospace software projects being tackled.” [289]

“During the past five to ten years, alternative, radically different life cycle models have been proposed, including rapid throwaway prototypes, incremental development, evolutionary prototypes, reusable software, and automated software synthesis.” [290]

“The rapid, throwaway prototype, made popular by Gomaa and Scott [8] in 1981, focuses on ensuring that the software product being proposed really meets the users’ needs.” [291]

“Incremental development [9] is the process for constructing a partial, but deployment-ready, implementation build of a system and incrementally adding increased functionality or performance.” [292]

“Evolutionary prototyping extends the concept of incremental development to its ultimate conclusion, viewing the software life cycle as a set of numerous prototypes that are evolved through successive experimentation and refinement to meet the user’s needs.” [292]

“Information sclerosis is a ‘syndrome familiar to operational information-based systems, in which temporary work-arounds for software deficiencies increasingly solidify into unchangeable constraints on evolution’ [4].” [293]

“The software industry is guilty of continuously reinventing the wheel. Reusability has achieved only limited application, mostly in business applications. There are few tools available to help reuse software designs or code from previous projects.” [293]

“The net effect of reusing components would be shorter development schedules (by using wheels rather than reinventing them) and more reliable software (by using components that have been previously “shaken down”).” [293]

“Transformational programming [16] is a methodology of program construction by successive application of transformation rules.” [294]

“It is difficult to compare and contrast these new models of software development because their disciples often use different terminology, and the models often have little in common except their beginnings (marked by a recognition that a problem exists) and ends (marked by the existence of a software solution).” [294]

“For every application beyond the trivial, user needs are constantly evolving.” [294]

“These metrics can later be used to compare and contrast sets of alternative life cycle approaches.” [295]

“The use of a rapid throwaway prototype early in the development life cycle increases the likelihood that customers and developers will have a better understanding of the real user needs that existed at time t0.” [296]

“Reuse of existing software components has the potential to decrease the initial development time for software significantly.” [297]

“The various life cycle alternatives reflect different approaches for improving the software development process.” [298]

**Disagreements**

“It is guaranteed that the final version of the program will satisfy the initial specification” [294]

This cannot be guaranteed. There is no “silver bullet” in the software world that can guarantee a process. All of these development models are designed to attempt to maximize success in the development process, but it would be impossible to guarantee complete satisfaction of requirements with Transformational Programming or any other model.

**Questions**

“Note that in the case of throwaway prototypes we are likely to implement only those aspects of the system that are poorly understood, but that in the case of evolutionary prototypes we are more likely to start with those system aspects that are best understood and thus build upon our strengths.” [293] I do not understand why this is so.