Software Process Models

IAN SOMMERVILLE

Computing Department, Lancaster University, Lancaster, UK ⟨is@comp.lancs.ac.uk⟩

The software process consists of the activities and associated information that are required to develop a software system. Every organization has its own specific software process but these individual approaches usually follow some more abstract generic process model. These generic process models are the subject of this article.

SPECIFICATION-BASED MODELS

The failure of several high-profile software projects in the 1960s led to the notion of a software life-cycle or process. The initial life-cycle model [Royce 1970] is now termed the waterfall model. This model consists of a set of phases, starting with system specification, with results cascading from one stage to another. The waterfall model includes the following phases:

(1) *Specification.* The functionality of the software and its operating constraints are specified in detail.
(2) *Design and implementation.* The overall structure of the software is designed and specific software components identified. These are implemented using some programming language, often by separate individuals or teams.
(3) *Integration and testing.* Individually developed modules are integrated into a complete system and tested.
(4) *Operation and maintenance.* The software is delivered to the customer and modified to meet changing requirements and to repair errors discovered in use.

The problem with this model is the lack of feedback from one stage to another. Specification, design, and implementation problems are often discovered only after implementation when the system has been integrated. Once a specification has been frozen, it is difficult to change in response to changing user needs. However, stakeholders in the system (end-users, managers, and so on) find it difficult to anticipate their real needs for software support, and both organizational and end-user requirements change during the development process. There is therefore a constant pressure for specification change.

This means that, in practice, there is always some iteration between the phases of the model, but this is invariably fairly limited and the delivered software may not meet the real needs of the customer. This led to widespread criticism of the waterfall model [Gladden 1982; McCracken and Jackson 1982] and the development of alternative software processes.

Incremental models [Mills et al. 1980] are a development of the waterfall model that attempt to provide some development stability while allowing users some opportunity for specification change. In this approach, the system functionality is partitioned into a series of increments and these are developed and delivered one by one. While one increment is being developed, its specification is frozen, but the specification of other increments may change. A version of the system is delivered early to users and they may experiment with it to help clarify their needs for later system increments.

The Cleanroom approach [Mills et al. 1987; Linger 1994] relies on incremental development, with each increment being formally specified. This specification is manually transformed into an implementation that is validated by formally proving that it meets its specification. There is no unit testing in the process and the objective of integration testing is to validate the system's reliability rather than discover system defects. This approach has reportedly led to a very low number of defects in delivered systems [Selby et al. 1987].

Specification-based models are most applicable to large system-engineering projects. Hardware and software subsystems may be developed in parallel by different organizations, and this requires both the system specification and the system architecture to be defined early in the development process.

EVOLUTIONARY DEVELOPMENT MODELS

The fundamental difficulties with a specification-based approach to development are that stakeholders in a system find it difficult to articulate their requirements in advance and that requirements change during the development process. An evolutionary approach to development integrates specification, design, and implementation. The stages in an evolutionary process are:

(1) Formulate an outline of the system requirements. This need be neither complete nor consistent but should give developers guidance as to what the system should do.
(2) Develop a system as rapidly as possible, based on this outline specification.
(3) Evaluate this system with users and modify the system until the system functionality meets the users' needs. This involves modifying the initial functionality of the system and adding new functionality as required.

This approach is usually supported by the use of domain-specific languages such as 4GLs or by very high-level, interpreted languages such as Prolog or Smalltalk. The developed system may be used as a basis for a system specification (throw-away prototyping) or may evolve into the system delivered to the user (evolutionary development).

While this development process is more likely to lead to software that is better suited to the *end-user's* requirements, it has a number of problems in its own right:

(1) It has an end-user focus, so critical organizational requirements (such as the need for interoperability) may not be given sufficient priority.
(2) The constant change to software degrades its structure so that the end result is often difficult and expensive to change. Consequently, the software is expensive to maintain and may have to be completely rewritten after a relatively short lifetime.
(3) The process does not have a high visibility and it is difficult for managers to assess how well development is proceeding. Many organizations are reluctant to use an evolutionary approach for large systems in which management is the principal problem.

In an attempt to integrate evolutionary development with the management requirements of large system engineering processes, Boehm [1988] proposed the spiral model of development. In this model, development spirals outward from a specification. In each round of the spiral, there is an explicit risk assessment and reduction phase. In this phase, prototype systems may be developed to reduce specification uncertainties, to establish a user interface design, and so on. These may either inform the system specification or evolve into the delivered system. Different parts of the system can be developed using different approaches.

Evolutionary development is the most appropriate approach for interactive systems with a significant user interface component and for innovative systems (for example, AI systems) whose requirements cannot be anticipated. This approach is also widely used for small and medium-sized business system development where it is supported by 4GLs.

CONCLUSION

The most appropriate software process model depends on the organization developing the software, the type of software being developed, and the capabilities of the staff. There is no "ideal" model and it makes little sense to try to fit all development into a single approach.

For large systems, a hybrid model is likely to be the most appropriate, where well understood parts of the system are developed using some form of the waterfall model, and those subsystems whose requirements are difficult to predict are developed using an evolutionary approach.

REFERENCES

BOCHM, B. W. 1988. A spiral model of software development and enhancement. *IEEE Computer, 21,* 5, 61–72.

GLADDEN, G. R. 1982. Stop the life cycle—I want to get off. *ACM Softw. Eng. Notes 7,* 2, 35–39.

LINGER, R. C. 1994. Cleanroom process model. *IEEE Software 11,* 2, 50–58.

MCCRACKEN, D. D. AND JACKSON, M. A. 1982. Life cycle concept considered harmful. *ACM Softw. Eng. Notes 7,* 2, 28–32.

MILLS, H. D., DYER, M. AND LINGER, R. 1987. Cleanroom software engineering. *IEEE Software 4,* 5, 19–25.

MILLS, H. D., O'NEILL, D., LINGER, R. C., DYER, M., AND QUINNAN, R. E. 1980. The management of software engineering. *IBM Syst. J. 24,* 2, 414–477.

ROYCE, W. W. 1970. Managing the development of large software systems: Concepts and techniques. In *Proceedings of IEEE WESTCON.* (Los Angeles, CA), 1–9.

SELBY, R. W., BASILI, V. R., AND BAKER, F. T. 1987. Cleanroom software development: An empirical evaluation. *IEEE Trans. Softw. Eng. SE-13,* 9, 1027–1037.