Introduction to Software Engineering Processes

Software Engineering Processes

Researchers at the Software Engineering Institute (SEI) (Carnegie Mellon University, 2006), and elsewhere, have often made the observation that following a defined process is more important than which process is followed. This week gives an overview of a few different processes that provide different perspectives on styles of processes.

A process represents the entire set of procedures and techniques used to guide a project. In this sense, a *methodology* is a part of a process that describes how to order tasks and determine which techniques are applicable for different tasks. A *method* then, is the specific set of tasks that are followed for a particular project. Some processes, such as OPEN (OPEN Consortium, 2006), include several methodologies that are applied to different types of projects.

In this course we will use the term *process model* to mean an abstract description of a generalised approach (or model) of developing software. A process then is a concrete implementation of a model that provides strong guidelines of how the process is meant to be followed. Using these definitions there are not many different process models but many different processes used by software developers. I have categorised process models as structured, incremental, prototypical, agile and formal. The term structured is used for those models that explicitly or implicitly aim for a single release of a final product; these models tend to be from earlier in software development history. Incremental models explicitly encourage a multi-staged release of a product and tend to be more recent than the structured models. The prototypical model describes those processes that are based on an exploratory approach to analysis and design and which assume uncertain requirements. Agile models focus on delivering value to the customer using a rapid learning and delivery model. The formal model is used to describe processes based on formal mathematical techniques.

All process models describe phases that occur in a development process; and all but the earliest models indicate that there is an iterative nature to the process. This means that problems or new information discovered in one phase will require a previous phase to be revisited, either extending or reworking the phase's products. This is not an unexpected aspect of process models as without iteration software development would have to proceed based on the assumption of perfect foreknowledge – which, given reality, would lead to software with known faults and that did not meet all of the client's needs.

Incremental process models advocate an approach to software development where a software system is built in a series of stages from a minimally functional shell, or partial system, to a fully integrated system. The intention of an incremental approach is to reduce risk in the development process. Risky and unknown aspects of the development can be tried first to discover if they are possible or if the risk is acceptable and if not the project can be cancelled early. An incremental approach also means that part of a system is delivered to the client early in the development process so that they can judge if it meets their needs and provide feedback to the developers.

Structured Models

Stagewise

The first published software engineering process was applied to the Semi-Automated Ground Environment (SAGE), an air defence project that started in the mid-1950s. This process has come to be known as the Stagewise process [Benington83], which was originally published in

June 1956. The Stagewise process describes a series of nine phases that occur during software development and places them in a hierarchical order in which the phases occur. These phases are Operational Plan, Operational Specifications, Program Specifications, Coding Specifications, Coding, Parameter Testing, Assembly Testing, Shakedown, and Evaluation.

Waterfall

The Waterfall process [Royce70] is the most widely known software engineering process. The Waterfall model elaborates on the Stagewise process. There is a different set of phases in the Waterfall process. The phases cover the same aspects of development as the Stagewise process but reflect changes in emphasis in the software industry. The phases are System Requirements, Software Requirements, Analysis, Program Design, Coding, Testing, and Operations. Commonly these steps are shortened to Analysis, Design, Implementation, Testing, and Maintenance [Sommerville01]. As shown in figure 1, the Waterfall process also extends the Stagewise process by introducing iteration between phases into the model. The original paper further points out that this iteration is likely to not just be to a previous stage but to possibly a much earlier phase. A commonly forgotten feature of the Waterfall model was that it originally advocated prototyping concepts. During the phases of Software Requirements, Analysis and Program Design an experimental version of the system was to be built. Program Design, Coding and Testing would then produce the complete system taking advantage of the lessons learnt from the experimental version. This was based on the experiences from the SAGE project and on the engineering practice of building a "model plant".

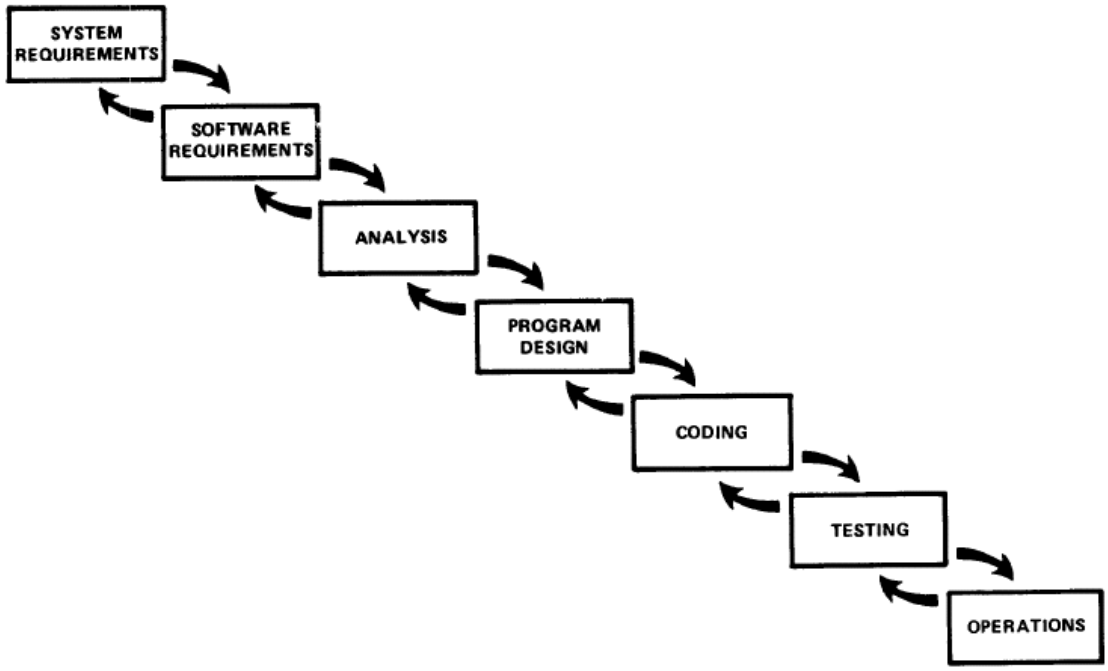

Figure 1 – Waterfall Model

Spiral

The Spiral process [Boehm88] differs from the other structured processes by not focusing on a sequence of phases but rather on controlling the development process within each phase. The process is depicted as a spiral, such as in figure 2. Each loop of the spiral represents a single phase of the process. The first loop is commonly equated to determining system feasibility. The next loop is commonly defining the requirements. The third loop is commonly system design. Fourth and further loops are usually associated with implementing the system. The graphical representation of the process in figure 2 indicates the activities that occur during each

part of a spiral. The spiral model can be easily extended to cater for an incremental approach to software development where more loops are added to the spiral to accommodate each incremental delivery.

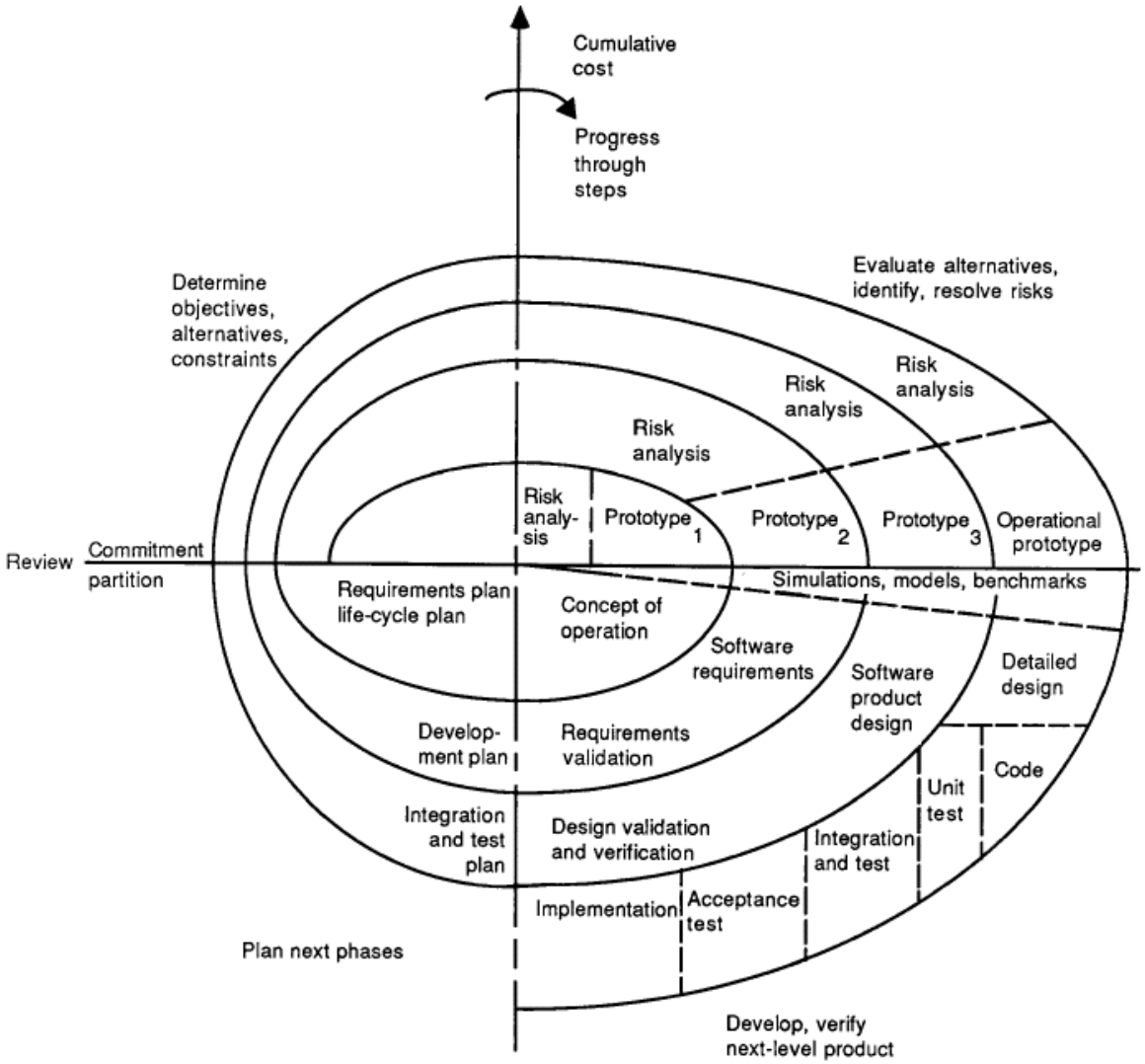

Figure 2 – Spiral Model

Each loop of the spiral is split into four quadrants Objective Setting, Risk Assessment and Reduction, Development and Validation, and Planning. Objective Setting defines specific objectives for that phase of the project and also identifies any risks to the project. Any constraints on the project or process are identified and a management plan is devised to enforce these constraints. Risk Assessment and Reduction takes each identified risk, analyses it and develops a plan to reduce the risk. Development and Validation chooses an appropriate development strategy for this phase and then implements that strategy. Planning first evaluates the results of the current spiral and if a decision is made to execute another loop of the spiral plans are specified for the next phase of the project. The project starts in the Objective Setting quadrant and terminates at the end of a Development and Validation quadrant. Maintenance is considered to be another project and so is not considered directly in the process.

Incremental Models

Unified Process

The Unified Process [Jacobson99] models the entire "life-span" of a system. During the life of a system it will undergo a series of cycles. Each cycle concludes with a product release to clients. Each cycle consists of four phases: inception, elaboration, construction and transition. Each phase is subdivided into some number of iterations. Inception is the starting phase where core requirements and business need are addressed. Elaboration refines the requirements in detail and develops a system architecture. The elaboration phase is intended to take two iterations. The construction phase is where the system is built in a series of iterative steps. Transition is when the system moves into beta testing and development focuses on fixing defects and deficiencies.

One point to note is that the Unified Process uses the terms iterative and incremental differently to their common English definitions. The Unified Process uses the term iterative to mean moving to another stage of development (what in English might be considered an incremental step). The term incremental is used loosely in the Unified Process with no clear definition.

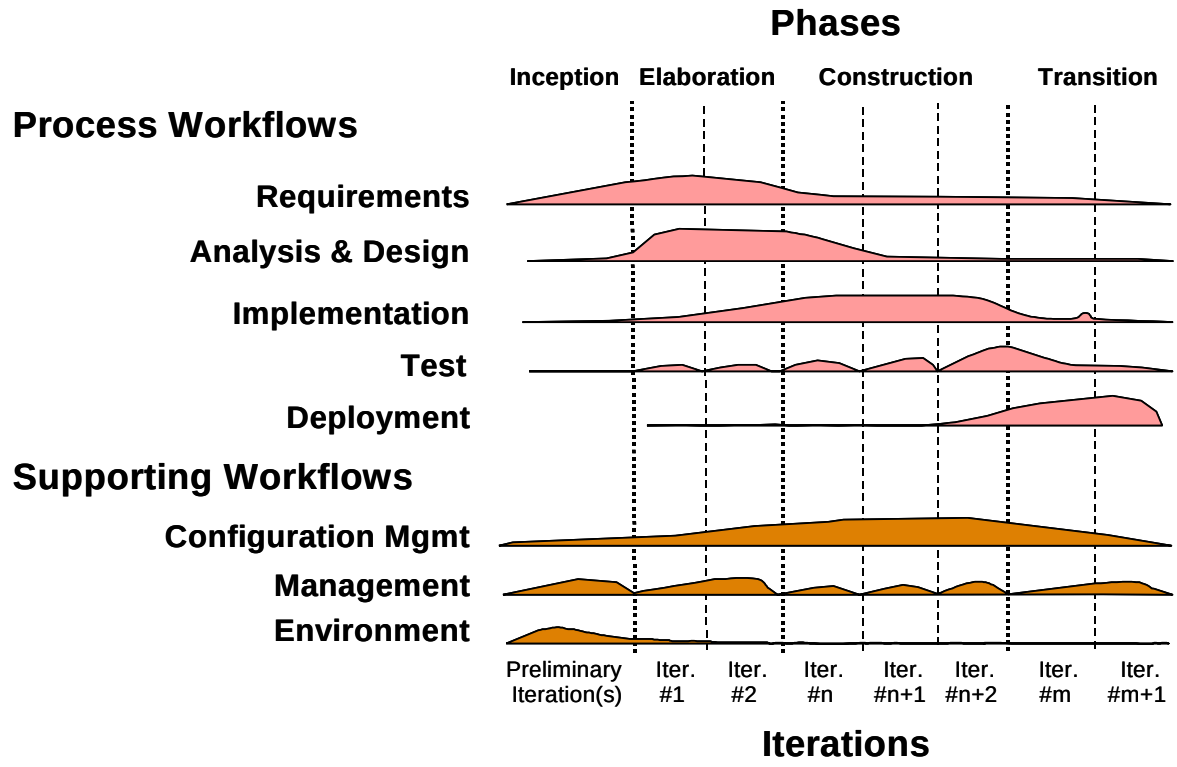

Figure 3 – Unified Process Model

OPEN

The Object-oriented Process, Environment, and Notation (OPEN) Process Framework (OPF) [Graham97] also supports the full lifecycle of a software system. OPEN is modelled by a set of metaclasses, which are clustered into five groups: work units, work products, producers, stages and languages. An OPEN compliant process will develop a process model based on the OPEN metamodel. To support process construction OPEN uses a contract model where process components have sets of pre and post - conditions governing their relationships.

Work Products are the artefacts or components developed by the project. Languages are the medium by which work products are described (e.g. UML notation, natural language, Z, source

code, …). Producers are those who create or maintain work products. Work Units are operations performed by a producer. These operations are defined as Activities, Tasks and Techniques. Activities are large-scale operations that tend to have "long-term" objectives. Tasks are "small scale" jobs that together make up an activity. A task is the smallest unit of work in the OPF and is used for fine-grained project management. Techniques are detailed descriptions of approaches to accomplish certain types of tasks. Stages are formally identified and managed time-periods in the project. The stages provide a macro view of the project schedule.

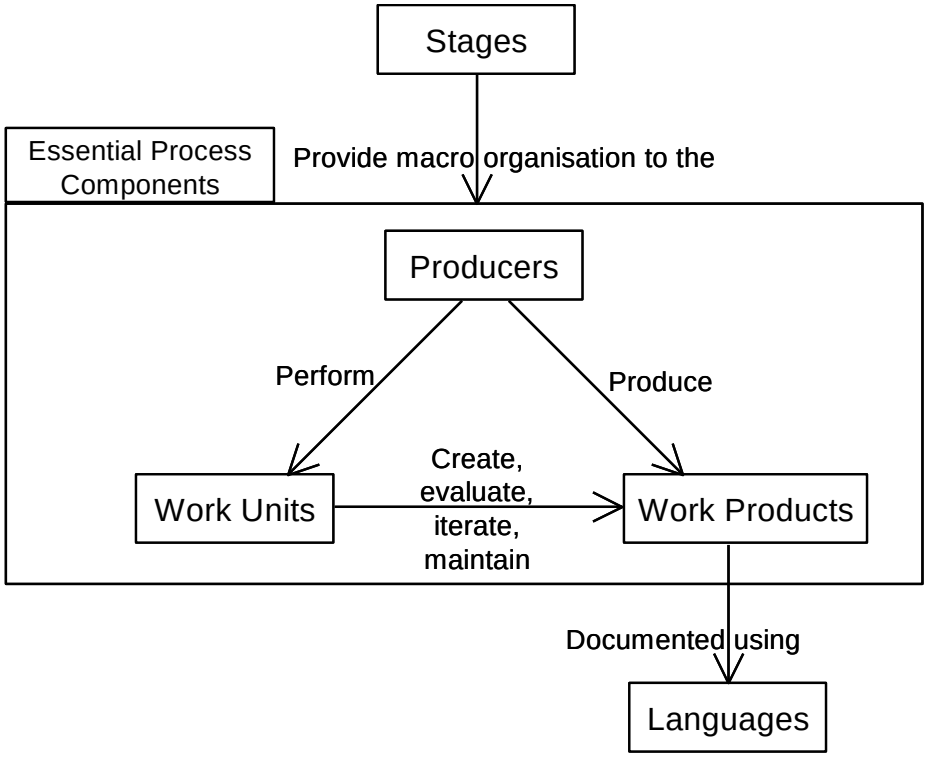

Figure 4 – OPEN Process Framework Model

Prototypical Model

Unlike the structured and incremental models there is no well documented model of the general prototypical approach. Instead there are a number of processes and methods that share the key characteristic of development based less on formal planning and more on development done in conjunction with a client to explore possible solutions. In this approach a general system description is used as the basis for developing an initial version of a system. User representatives evaluate this system and the results of this evaluation are fed into the redevelopment of the prototype. This cycle is repeated until an acceptable system is delivered or the project is cancelled. A prototyping approach is predicated upon the assumption that each prototype can be developed and delivered quickly. This type of development is very useful when there are no clear requirements for a system and allows various options to be explored. Rapid Application Development (RAD) [Millington95] and Joint Application Development (JAD) [Wood95] are two of the original processes based on the prototyping model.

Agile Model

Agile methods have recently become a popular approach to software development. There are several different agile processes and methods such as Scrum [Schwaber02], Extreme

Programming [Beck04], Crystal [Cockburn02], Dynamic Systems Development Method (DSDM) [Stapleton97], Feature Driven Design [Coad00] and Adaptive Software Development [Highsmith00]. These processes share with the prototypical model the characteristics of frequent delivery of software, close interaction between clients and developers, and the expectation that requirements will change. Two differences to some prototyping processes are that the system is evolved from the start, there is no "throw away" prototype (this is some times referred to as "evolutionary prototyping"), and requirements change is encouraged even near the end of the development process. Beyond the core prototyping characteristics agile methods share the characteristics of assuming developers are competent enough to work with minimal supervision, relying on face-to-face communication, focussing on the technical quality of the software, focussing on simplicity or minimalist design, the belief that the best designs evolve from team interaction and not through planning, producing minimal documentation outside of the code, and reflection on the development process [Agile Alliance01].

Formal Model

Like the prototypical model there is no well documented model of formal system development, but there are a few processes and methods that share mathematical formality as their core practice. Formal processes and methods share the belief that software development is an inherently logical activity and is best conducted by application of formal logic. This is typically conducted by developing a formal specification for a system using a formal language such as Z [Spivey92], VDM [Dawes91] or B [Abrial96]. This specification is then transformed into code through the application of mathematical rules in a series of stages. The most widely known formal process is the Cleanroom process [Mills87]. In this process the software system is initially formally specified. An incremental development approach is used where each part of the system is developed and validated separately. The specification is refined through a series of small formal transformations to end up with the code based on simple structured programming constructs. The software is then statically verified through a rigorous inspection process using mathematical arguments rather than formal proofs. Lastly the system undergoes statistical reliability testing. An interesting aspect of this process is that in its pure form no dynamic testing is conducted until system integration. Three "teams" are used in the Cleanroom process. These are the specification team, development team and certification team. The specification team develops the client-oriented and formal specifications for the system. The development team transforms the formal specification into the software and inspects the software. The certification team develops the dynamic system tests based on the formal specification. The software development and test development occur in parallel. Thus it is common that the specification team personnel will become part of either the development or certification teams. There is some evidence to support the clam that the Cleanroom approach is just as cost-effective as non-formal processes and that it leads to software with fewer defects [Cobb90].

Hard versus Soft Methodologies

Some people try to distinguish between hard and soft methodologies. A *hard methodology* is based on engineering principles and is prescriptive in what should be done. A *soft methodology* is mainly concerned with the people involved in the development and focuses on providing the best outcome in a social context. Modern successful development processes usually incorporate aspects from both hard and soft methodologies. The structure and guidance provided by hard methodologies can be complemented by consideration of the human environment into which the system will be deployed. (For an interesting extension on the soft methodologies ideas

consider Christopher Alexander's keynote address at OOPSLA'96 [1996], particularly section C on the "Generation of a Living World".)

References

Abrial, Jean-Raymond. *The B Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

Agile Alliance, The. (2001) *The Agile Development Manifesto* [Internet]. Available at: http://agilemanifesto.org/.

Alexander, C. (1996) *The Origins of Pattern Theory, the Future of the Theory, and the Generation of a Living World* [Internet]. Available at: http://www.patternlanguage.com/archive/ieee/ieeetext.htm.

Beck, Kent. *Extreme Programming Explained: Embrace Change*, 2nd edition. Addison-Wesley, 2004.

Benington, Herbert. "Production of Large Computer Programs." *Annals of the History of Computing*. ACM, Oct. 1983, pp. 350 - 361.

Boehm, Barry. "A Spiral Model of Software Development and Enhancement." *IEEE Computer*, Vol. 21, No. 5. IEEE Press, May 1988, pp. 61 - 72.

Carnegie Mellon University. (2006) *Software Engineering Institute* [Internet]. Available at: http://www.sei.cmu.edu/.

Coad, P., E. LeFebvre and J. De Luca. *Java Modeling in Color with UML: Enterprise Components and Process*. Prentice Hall, 2000.

Cobb, R. and H. Mills. "Engineering Software under Statistical Quality Control." *IEEE Software*, Vol. 7, No. 6. IEEE Press, 1990, pp. 44 - 54.

Cockburn, Alistair. *Agile Software Development*. Addison-Wesley, 2002.

Dawes, John. *The VDM-SL Reference Guide*. Pitman, 1991.

Graham, Ian, Brian Henderson-Sellers and Houman Younessi. *The OPEN Process Specification*. Addison-Wesley, 1997.

Highsmith, Jim. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House Publishing, 2000.

Millington, D. and J. Stapleton. "Special Report: Developing a RAD Standard." *IEEE Software*, Vol. 12, No. 5. IEEE Press, Sept. / Oct. 1995, pp. 54 - 56.

Mills, H., M. Dyer et al. "Cleanroom Software Engineering." *IEEE Software*, Vol. 4, No. 5. IEEE Press, Sept. / Oct. 1987, pp. 19 - 25.

OPEN Consortium. (2006) *Object-Oriented Process, Environment and Notation (OPEN)* [Internet]. Available at: http://www.open.org.au/.

Jacobson, Ivar, Grady Booch and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

Royce, W. W. "Managing the Development of Large Software Systems: Concepts and Techniques." *Proceedings of IEEE WESCON*. IEEE Press, 1970, pp. 1 - 9.

Schwaber, K. and M. Beedle. *Agile Software Development with SCRUM*. Prentice Hall, 2002.

Sommerville, Ian. *Software Engineering*, 6th edition. Addison-Wesley, 2001.

Spivey, J. M. *The Z Notation: A Reference Manual*. Prentice Hall, 1992.

Stapleton, J. *Dynamic Systems Development Method - The Method in Practice*. Addison-Wesley, 1997.

Wood J. and D. Silver. *Joint Application Development*, 2nd edition. Wiley, 1995.

