Journal of Universal Computer Science, vol. 10, no. 4 (2004), 326-358
submitted: 17/10/03, accepted: 2/2/04, appeared: 28/4/04

Process Construction and Customization

Brian Henderson-Sellers
(University of Technology, Sydney, Australia
`brian@it.uts.edu.au`)

Magdy Serour
(University of Technology, Sydney, Australia
`mserour@it.uts.edu.au`)

Tom McBride
(University of Technology, Sydney, Australia
`mcbride@it.uts.edu.au`)

Cesar Gonzalez-Perez
(University of Technology, Sydney, Australia
`cesargon@it.uts.edu.au`)

Lorraine Dagher
(University of Technology, Sydney, Australia
`ldagher@it.uts.edu.au`)

Abstract: Adopting the most appropriate methodology for particular software developments remains a challenge for all industrial IT organizations. Previous attempts to promote a single approach as useful for *all* occasions has proven untenable. Rather, a combination of a metamodel and a repository of process/method components ("method engineering") provides a more efficacious approach, particularly as elements of the method engineering approach are able to be automated. In this paper, we advocate the use of method engineering, illustrating its utility by the construction of methodologies at various levels of process capability.

Key Words: design methods, design, software process
Category: D2.10, K6.3

1 Introduction

Information systems for commercial application need to be of high quality, flexible and maintainable. Building such systems needs some sort of process. The process an organization uses to build software applications can vary enormously [Cockburn, 2000]. Furthermore, different organizations can have different capability to execute such a process. There are different levels of process capability for an organization, as exemplified by, for instance, CMM [Paulk et al., 1993] and SPICE (ISO 15504) [ISO, 1998]. For organizations aiming to achieve a CMM/ SPICE level 3, having a repeatable process is critical. Such a process needs to be able to assist in optimizing a development team's performance in creating a high quality software product. While there are many processes available, few have the ability to be customized to particular situations and particular capability levels. There is a general "one size fits all" mindset in these software methodologies. Flexibility is limited, which can lead many organizations into dysfunctional practices.

In this paper, we examine the current problems with software development methodologies, first in terms of currently available support [see Section 2] and then in terms of the importance of quality and productivity [see Section 3]. We advocate extensive adoption of method engineering and, more specifically, situational method engineering (SME) as a means of providing at least a partial solution to today's concerns of poor productivity and poor software quality [see Section 4]. Finally, we present a summary of conclusions and some relevant bibliography [see Section 5 and References].

2 Limitations of Existing Processes

Recently, several research groups have identified the need to modify or customize software processes [Pérez et al., 1995] [van Slooten and Hodes, 1996] [Chroust, 2000] [Ralyté and Rolland, 2001]. In the pre-OO process environment, the TAME project aimed to deliver a tool to be used to permit a more quantitative and objective assessment of the optimal tailoring required to improve a process [Basili and Rombach, 1987]. The SPEARMINT approach [Scott et al., 2001] provides a means of documenting and modelling existing processes, and the PIE methodology [Cunin et al., 2001] addresses issues relevant to the modelling and management of processes as they evolve during projects. The aim of making process construction more rigorous is also described in the process construction kit of [Rupprecht et al., 2000] and the process framework of [Hruby, 2000]. More recently, the notions of process diversity were espoused in a special issue of IEEE Software (July/August 2000) in which [Glass, 2000a] notes the recent growth in the method engineering community.

The term process may be defined in two slightly different ways. The software development community (here our primary focus being the object-oriented community) tend to use process and methodology typically as synonyms to mean everything that needs to be done in a successful development. For example, Breton and Bézivin describe it as "a set of work items, scheduled according to constraints, which all participate in fulfilling a common purpose" [Breton and Bézivin, 2001]. This use of the appellation "process" tends to include human resource issues, technology issues, lifecycle issues as well as issues pertaining to software development "phases" [Henderson-Sellers et al., 1998]; although when qualified it may be used on a smaller scale, e.g., the "software maintenance process" [Breton and Bézivin, 2001]. In contrast, the capability assessment and standards communities tend to use the term process at a smaller scale, as equivalent to terms like Activity in OPEN [Graham et al., 1997] or Discipline in SPEM [OMG, 2003]. If the term process is defined as a transformation from input into output [ISO, 1998], the notions of process (improvement) in many ISO standards and Software Process Improvement contexts (SPI) [Jalote, 2002] underline that the granularity of interest is often called method fragment [van Slooten and Hodes, 1996][Brinkkemper et al., 2001], method chunk [Ralyté and Rolland, 2001], or, very intuitively, process component [Henderson-Sellers et al., 2002][Firesmith and Henderson-Sellers, 2002].

We should also note that the terminology of process, method and methodology is often highly confused. The term "methodology" is usually understood to have two components: process and product [Rolland et al., 1999]. In this paper, however, we will only be addressing the process component of the methodology and so will take the liberty of using process and methodology as synonyms (also the term "method" is taken as a synonym of methodology).

There are currently several object-oriented software development methodologies, processes, or methods available for developing software applications. They are presented in such a way as to assist developers in producing high quality and robust software on time and on budget to satisfy the organization's business needs. Software methods have a responsibility to facilitate creativity and, at the same time, provide an ultimate way for planning and managing every phase of the entire software development lifecycle for different project, organization and even different domain constraints. Contemporary OO software development processes range in "size" from "light-weight" or "agile" to "heavy-weight" or "formal," broadly falling into two categories.

The first category can be best described as well-established and large-scale processes. The second category covers those processes that advocate a new style of software development, often described as agile or light-weight processes. They are based on the enforcement of embracing and responding quickly to changes in a fast, iterative, incremental manner. eXtreme Programming [Beck, 1999], Crystal [Cockburn, 2001] and SCRUM [Beedle et al., 2002] are examples of these agile processes. Agile methods have, to date, been largely trialled on small projects and there is significant debate as to what extent they will or will not scale up (see, for example, [Constantine, 2002]). The agile community strongly claim that agile/lightweight methods are seen to be more people oriented and more able to adapt to changes than the more traditional approaches.

In general, software methods offer good advice and focus on the entire software development life cycle. On the other hand, they do not usually offer a great deal of flexibility or tailorability to enable and empower a development team to adapt them to best suit a particular project.

Constantine and Lockwood claim that it is not possible to design or select a single method for all conceivable situations, especially the prevailing organizational culture [Constantine and Lockwood, 1994]. Most of the contemporary methods purport to offer "the solution" to almost all software projects by direct adoption with a low degree of customization. The formal, so-called "heavyweight", methods, both in structured development and, recently, in object-oriented development, have been described as inflexible and hard-to-follow methods. [Fowler, 2002] argues that the existing/dominant heavyweight methods have too much overhead and are resource hungry. Indeed, many authors cite the popularity of the CMM (and now the CMMI) and ISO12207 as leading to organizations creating and using processes that are more "heavy" than necessary.

Existing "heavyweight" methods tend to be predictive and unable to react to changing circumstances. They are thus useful for fixed price and/or fixed scope contracts, particularly where requirements are well-known, and can be used when significant project management overheads must be incurred, e.g., with large, business-critical systems built by teams of over 50 members [Fowler, 2002].

One current question now is, have agile methods fixed the major drawbacks and removed the weaknesses of the formal methods? Our current research suggests that agile/lightweight methods impose some significant amount of rigour on the process component of the method. Henninger *et al.* assert that when moving from project to project in different domains, with varying project size or scope, for instance, it is unlikely that the same agile method will be appropriate [Henninger et al., 2002]. With XP, for example [Beck, 1999], there are a number of pre-specified practices and values. Advocates of XP state publicly that if any one of these is missing then it is inappropriate for the development team to claim they are using XP. In other words, while the method itself provides a high degree of flexibility to react to customers' whims, the overall project management element and the very structure of the procedures tend to be tightly controlled (as indeed they must be if such a rapid, high pressure development can succeed.)

In addition, there are a number of recent research efforts that aspire to add more values to the existing agile methods in order to reduce their weaknesses. Some examples can be found in [Patton, 2003a][Abrahamsson et al., 2003] and, most recently, in the integration of Usage Centered Design (UCD) into agile methods [Patton, 2003b].

To sum up the above aphorisms, neither formal nor agile methods satisfy and fulfil the ever-increasing demands of the software development community. For many situations, heavyweight methods fail to offer a sufficient degree of flexibility and tailorability to be customized for different projects. On the other hand, existing agile methods do not offer any flexibility whatsoever that enables developers to self tune or re-engineer their method to suit a different project, organization or even a different domain.

2.1 Criteria for Process Flexibility Assessment

The usage of a process by an organization is, by its nature, a delicate and unstable situation. On the one hand, the process must guide the organization in the appropriate direction, compelling it to "do the right thing". On the other hand, the organization must find its particular skills and values supported and respected by the process rather than being exhorted to use unfamiliar techniques or approaches. Most of the processes available today promise guidance and guarantee superior results if you follow them but make no claims with regard to considering the organization's existing assets. Therefore, the successful adoption of a particular process is, in real life, only possible if the organization accommodates a large number of changes dictated by a pre-defined, rigid process. For this reason, flexibility becomes an important and desirable characteristic of a process. However, process flexibility cannot be approached in an unstructured way or as an afterthought. The process must be flexible (so it can accept each organization's work style and skills) but also, regardless of how it is customised, maintain an appropriate degree of formalism and efficacy.

In order to assess existing processes with regard to their flexibility and capability for customization, some objective criteria must be developed. Organizational variability, the reason behind the need for process customization, can provide some insights to these:

- Different organizations possess varied degrees of expertise in different skills and techniques. Organizational culture and style often help determine which techniques are used and which are not.
- Different projects (even within the same organization) often demand different degrees of formality and accuracy, depending on their criticality and quality requirements. Not all projects require the same tasks to be done and products to be developed.
- Different organizations often perform at different capability or maturity levels. Usually, organizations performing at higher capability levels carry out jobs (such as process measurement or detailed planning) that organizations performing at lower capability levels simply omit.

Some criteria can be derived from these facts:

Criterion A: Does the process allow for technique selection for each task?

Criterion B: Does the process allow for the selection of what activities, tasks and techniques are to be used in a per-project basis, depending on the specific characteristics of the project?

Criterion C: Does the process allow for the customization of what activities and tasks are performed, and what work products are constructed, depending on the capability level of the organization and/or development team?

Criteria A and B show that different techniques may be used to achieve the same results. For example, a wide range of techniques can be used to develop a system requirements specification. Some organizations and some projects might make use of more formal techniques, such as questionnaires, workshops and joint application development, while others would use storyboarding and throwaway prototyping. A process capability to allow an organization to decide on the techniques to use in order to realise a given task or produce a given product is, therefore, the first criterion to assess the process' flexibility.

Criterion B also illustrates that projects of different kinds have different needs in terms of the products, i.e., models and documents, that must be created and the tasks that must be executed. For example, the development of a class library that lacks a user interface will not need to create user interface diagrams and will not need to perform tasks related to usability assessment or the definition of user navigation paths. A project aiming to produce a business application with a complex user interface, on the other hand, will need to spend a considerable amount of time on these issues. A process' capability to allow an organization to selectively select or omit portions of the process is the second criterion for process flexibility assessment.

Finally, Criterion C shows that some tasks (and, consequently, some intermediate products) may or may not be performed during a project depending on the capability level at which the organization is performing. For example, an organization performing at CMM level 2 or above will want to perform some kind of quality management activities (and for this reason generate products such as quality plans and quality assurance records), while an organization performing at level 1 will completely exclude such quality assurance chores. Sometimes, this difference is more subtle, affecting not whether an organization performs a whole activity (such as quality management or requirements engineering) but which specific tasks it will perform within a given activity. For example, a level 1 organization will perform requirements engineering by analysing stakeholders and users, developing a vision statement, obtaining and analysing the requirements and then preparing a specification document. A level 2 organization will perform all of these tasks as well, but will also develop a plan for the whole activity and verify all the generated work products for correctness. A level 3 organization will add process measurement and improvement tasks to these, and so on. A process' capability to allow an organization to add or remove details on the jobs to be performed depending on the required capability level is, therefore, the third criterion for process flexibility assessment.

2.2 Assessment of Existing Process Approaches

There are many processes available currently, but few can be customised to a satisfactory degree. Using the three flexibility assessment criteria developed in the previous section, it is possible to objectively evaluate how flexible processes are. We will focus on the following processes or process frameworks/metamodels, which are (in alphabetical order): Catalysis [D'Souza and Wills, 1999], CMMI [SEI, 2000], Extreme Programming or XP for short [Beck, 1999], ISO/IEC 12207 [ISO, 2002], ISO/IEC 15504 [ISO, 1998], OMG SPEM [OMG, 2003], OOSPICE [Stallinger et al., 2003], OPEN [Firesmith and Henderson-Sellers, 2002], and the well-known Rational Unified Process (RUP) [Kruchten, 1999][MacIsaac, 2003]

[Table 1] shows how these processes or process frameworks score in relation with each of the defined flexibility assessment criteria. Details for each process or process framework are given below.

Catalysis: it makes no distinction between techniques (how to achieve a specific result) and tasks (what is expected to achieve), offering process patterns that can be freely

Process or framework	**Criterion** A **Task customization**	**Criterion** B **Project customization**	**Criterion C Maturity customization**
Catalysis	No	Yes	No
CMMI	n/a	No	Yes
Extreme Programming	No	No	No
ISO/IEC 12207	n/a	Yes	No
ISO/IEC 15504	n/a	Yes	Yes
OMG SPEM	Yes	Yes	No
OOSPICE	Yes	Yes	Yes
OPEN	Yes	Yes	No
RUP	No	Yes	No

Table 1: Degree of compliance of each of the studied processes/frameworks with each of the three objective criteria

customised. Therefore, no support for technique-to-task mappings exists. However, Catalysis process patterns comprise an original approach to process fragment selection, allowing developers to use the patterns that they consider most appropriate and ignoring others. Finally, Catalysis makes no mention of capability levels.

CMMI: it is not a process or a process framework in itself, but contains a process reference model used to perform process assessments. Therefore, the first criterion in our list cannot be evaluated. With regard to process fragment selection, CMMI approaches assessment from a monolithic point of view, so no discrete process fragments are considered. Finally, capability levels are considered by CMMI.

Extreme Programming: together with other agile approaches to software development, it is extremely rigid in its prescriptiveness of tasks and techniques to be used. No flexible mappings are possible, since all the techniques to be used are predefined by the method itself. Also, leaving out a fragment of the process is explicitly forbidden, and there is no mention of capability levels.

ISO/IEC 12207: it uses abstract definitions of the jobs to be performed, so no support for technique-to-task mappings is given. Discrete process fragments are defined, so selection facilities exist. Finally, 12207 makes no reference to capability levels.

ISO/IEC 15504: similarly to CMMI, is not a process framework but an assessment standard that contains a process reference model, so the first criterion in our list cannot be evaluated. However, since 15504 addresses process assessment using discrete process fragments, selection facilities are considered to be present. Finally, 15504 makes explicit use of capability levels.

SPEM: it provides a Guidance class (of which Technique is a kind) that can be mapped to other model elements in a many-to-many fashion, so a flexible technique-to-task mapping can be achieved. Also, SPEM provides for the definition of discrete activities and steps, so process fragment selection facilities do exist. Finally, capability levels are not considered by SPEM.

OOSPICE: it has been designed with flexibility criteria in mind, so it satisfies all of them. OOSPICE includes a TaskTechniqueMapping class that allows flexible mapping between tasks and techniques. Also, it approaches the definition of a process using discrete chunks that can be selected depending on the needs. Finally, OOSPICE incorporates minimum capability level attributes for some of its classes, so processes can be tagged with the appropriate capability level as necessary.

OPEN: it provides a deontic matrix to map tasks to techniques, so a flexible mapping is easily attained. Also, process fragment selection is supported through the OPEN repository of process components. Finally, OPEN does not directly support the concept of capability level.

RUP: it does not provide a mechanism to allow mapping techniques to tasks, since process components are provided as pre-packaged chunks. However, process fragment selection is supported through such packages. Finally, no mention of capability levels is made.

[Table 2] summarises the scores for each of the evaluated process or process framework. A value of 1 has been assigned to each "yes" in the previous table, and a value of -1 has been assigned to each "no". The result of this numerical conversion is shown in the column titled "Score". The column "Normalized Score" shows the result of normalizing the scores to a $[0, 100]$ interval. It can be readily observed that the "processes" with the best scores (100 on the normalized scale) are those associated with a metamodel-based process focusing on capability assessment (OOSPICE). Those with the lowest scores are those created as one-off methodologies/processes (Catalysis, XP and RUP), with neither capability nor explicit metamodel content, i.e., no notion of method engineering. Both metamodel-focused methodological frameworks, OPEN and SPEM, have the same score, in the upper half of the scale.

3 Importance of problem: Improving productivity and quality

The software industry, although improving, is plagued by overruns in budget, poor quality and cancelled projects [Yourdon and Argila, 1996], sometimes colloquially known as the "software crisis" [Onoma, 1987][Laplante, 1998]. While [Glass, 2000b] agrees that there are software project failures, he argues that this does not necessarily indicate a crisis. His view is that reports of failed software projects, especially large ones, are exception reports and newsworthy because of the rarity rather than a general comment on the state of software development. Glass acknowledges that other industry commentators do not share his view and are more pessimistic. The Chaos

Process or Framework	Score	Normalized Score
Catalysis	-1	33
CMMI	0	50
Extreme Programming	-3	0
ISO/IEC 12207	0	50
ISO/IEC 15504	2	83
OMG SPEM	1	67
OOSPICE	3	100
OPEN	1	67
RUP	-1	33

Table 2: Flexibility scores for the studied processes/frameworks

Report [Johnson et al., 2001] similarly shows a steady improvement in the number of projects that complete.

In 1994, only 16% of application development projects met the criteria for success (completed on time, on budget, and with all features/functions originally specified.) In 2000, 28% of projects were in the successful column. In 1994, 28,000 projects were successful, while in 2000, the number rose to 78,000, e.g., almost a threefold increase. Conversely, failed projects amounted to 54,000 in the 1994 study as compared to 65,000 in the 2000 study. This was an 18% increase, while overall project growth exceeded 60%. Still, 65,000 is a large number.

Richard Hunter from the Gartner Group [Hunter, 1998] asserts that consistent adherence to a moderately rigorous software methodology/process can provide 70% of IT organizations with a productivity improvement of at least 30% within two years. This fact emphasises the need for inserting a formal software development process in practice. A proper process intends to make software development more formal, rigorous and repeatable as far as it is possible. Finally, development knowledge should not remain in the brain of a single person, but in an organization's documentation.

Measures of software project success, particularly those used in long lived comparisons such as the Standish Group's Chaos Report [Group, 1994], have generally been measured on completion, budget, schedule and accepted functionality. Quality levels, i.e., an absence of defects, are judged indirectly by the software being of sufficient quality to have been accepted. The International Software Benchmarking Standards Group [ISBSG, 2003] measure productivity very specifically by the product delivery rate, measured as the number of hours per function point. There is no measure given for quality or defect density. Again, it is assumed that the measure of quality is that the delivered product was of sufficient quality to be accepted. The desire to improve software productivity and quality can stem from a desire to compete in the market place just as strongly as a desire to avoid an expensive failure. When software systems are acquired rather than developed in-house, it is natural to seek some assurance that the supplier has their development processes under control, will deliver to budget, schedule and with the required functionality. Market forces will tend to favour those with higher productivity and quality (sufficient motivation in itself to seek better software development practices.) Obviously those with more productive processes can submit a lower bid for the same work, or attract a better profit. From the other side of the bargaining table, a customer may demand that an organization carry out some process improvements as a contractual condition.

There is a general antipathy to adopting a process, often being seen as a "waste of time". A study by Baddoo and Hall found that the most demotivator against process improvement was commercial pressure [Baddoo and Hall, 2003]. Making improvements takes time and will generally reduce productivity as the developers learn the new skills or techniques [van Zyl and Walker, 1982]. That the same improvements will ultimately result in considerably higher productivity is not an effective answer to a demanding customer concerned about their specific deadlines. Often the process improvements fall victim to such commercial pressures. Baddoo and Hall also found that resistance to software process improvement was largely due to inertia, bad experience and general lack of support for software process improvement (SPI). Inertia stems from a comfort with the familiar and an unwillingness to adopt a new, and unfamiliar, process. These were both coupled with a lack of managerial support throughout the improvement process.

Time to market dominates thinking, often totally at the expense of the need for quality. Sometimes there is good reason why an application must be completed by a given deadline. When completion is timed to coincide with another event, such as the completion of a building or a tunnel or a factory, there is a very real need to meet a project schedule. However, when the product is intended for a general market, the schedule pressure is more to match or beat competitors.

Software development is often seen as a production process rather than a design or problem solving process. That it, the problem to be solved is relatively known and developing a system that implements the solution is likened to the production of something.

Production that is less frequently repeated in the sense of, for example, building construction is characterized by extensive planning to reduce, if not remove, unknowns from a schedule followed by execution of the scheduled activities in pursuit of the final goal [Muller, 1982][Yates and Tatum, 1982]. The expression "plan the work then work the plan" neatly sums up this approach. Essential to this type of planning is that the techniques to be used are well known and the problems to be solved have precedent solutions. It is the rare buildings such as the Sydney Opera House that challenges existing construction processes.

Such an approach assumes that all problems are solvable if they are decomposed sufficiently, and that the solution can be reassembled from the component solutions, i.e., the whole is a sum of its parts. Many articles on project management reflect this model, for instance: [Bauch and Chung, 2001] [McConnell, 1998] [Scarola and Tatum, 1982]

[Hughes and Cotterell, 1999] [ISO, 1999] [Thomsett, 2002] [Boehm et al., 2000] or [PMI, 2000] or are good examples. A system will be divided into components and sub-components with the expectation that each sub-component problem can be solved, and that the resultant components will interface and interact in predetermined ways. The whole becomes the sum of its parts.

[Gasson, 1998] identifies "a traditional, rational model of design [that] nominates most current organizational information systems development practice. This model is based on the conceptualization of IS development as a staged, linear, decompositional process." Projects treated in this way place great emphasis on planning and estimation. The assumption is that planning, with only minor corrections, will be done once and will accurately predict how the project will be carried out. Naturally, this requires detailed estimates of how long each task will take and the resources required for its completion. With the software development project divided into component parts in this way, it becomes possible to treat software development as an engineering process, amenable to many of the same monitoring, management and improvement techniques [Humphrey, 1989][Humphrey, 1994][Humphrey, 2000].

Unfortunately, no metrics exist to demonstrate that such myopic thinking leads frequently to loss of opportunities and/or defunct businesses, although there is a growing body of evidence to suggest that higher maturity levels are associated with better software development performance [Lawlis et al., 1995][Brodman and Johnson, 1996]. In contrast, if software development is viewed as a design process, then there is an emphasis on solving the problem at hand, rather than implementing an already known solution. In design problems, people have to collaborate and social processes are essential [Gruhn, 1992]. [Gasson, 1998] argues that the design activity cannot be separated into a single stage of the system development life cycle. Gasson further argues that the traditional model requires that all the requirements are defined before problem decomposition begins. Obviously this is seldom true when, on average, only 58% of requirements were specified before beginning product design [Thomke and Reinertsen, 1998].

Major benefits that we, the software community, can achieve from deploying a formal software process in practice are as follow:

- Ensuring that a consistent, reproducible approach is applied to all projects in the way of providing a uniform approach to developing software.
- Achieving a clear definition and clear management of each phase with its pre- and post-conditions.
- Overcoming the inadequacy and imperfection of the ad hoc process.
- Controlling and supporting the whole development life cycle and providing good project planning and management.
- Gaining consistency and traceability through the whole development life cycle.

- Emphasizing the analysis and understanding of the problem through customer involvement.
- Supporting a whole suite of projects (a “programme”).
- Reducing the risk associated with shortcuts and honest mistakes.
- Simplifying the complexity of today’s requirements and best handling the enormous size by decomposition to manageable pieces of software.
- Producing complete and consistent documentation from one project to the next.
- Gaining a degree of flexibility to tailoring the process to manage and support different types of projects and strategies.

In summary, following a formal OO Software Engineering Process (SEP) through the full lifecycle of software development can maximize and facilitate the benefits of producing high quality software, including interoperability, portability (platform independency), adaptability, scalability, supporting graphical user interfaces and providing better fault tolerance and more network management.

3.1 Software Process and CMM

The SEI Capability Maturity Model for Software (CMM) [Paulk et al., 1993] was developed by the Software Engineering Institute (SEI), to set a standard and provide guidance for developing software engineering disciplines and management. It describes a framework that organizations can use to determine their ability to develop and maintain software and is a model for organizational improvement. Its use is becoming widespread among software developers, especially for companies developing large-scale software in a competitive procurement environment. Government and corporate software customers are increasingly requiring that proposals include information about a software development organization’s level of certification of the CMM. SEI has defined five levels to characterise the maturity of a software development organization, as follows:

Level 1 (Initial): Processes are ad hoc and occasionally chaotic. Few processes are defined. Success depends on individual effort and heroics.

Level 2 (Repeatable): Basic project management processes are established to track cost, schedule and functionality. A process discipline is in place to repeat earlier successes on projects with similar applications.

Level 3 (Defined): Management and engineering processes are documented and integrated into a standard software process. Projects use an approved, tailored version of the organization’s standard software process.

Level 4 (Managed): Detailed measures of the software process and product quality are collected. Processes and products are quantitatively understood and controlled.

Level 5 (Optimizing): Continuous process improvement is aided by quantitative feedback from the process, and from piloting innovative ideas and technologies.

3.2 OO Process

An object-oriented process provides the step-by-step activities that lead from analysis to implementation. The process provides a set of graphical notations for use in reviews, inspections and documentation, and also provides communication techniques to bridge the gap between software developers and customers.

An OO process provides a communication medium between developers and customers, a framework for modelling the domain problem, and standards for transitioning the problem from analysis to final deliverable products. The process has an impact on almost every step in the software development activities.

An OO process provides predictability, repeatability and documentability. Software systems without documentation can be very costly to enhance and maintain. Each enhancement or maintenance task will cost as much or more than the original development cost.

An OO process provides a way for project managers to successfully manage their projects, as well as a way for all system stakeholders to monitor the project progress.

3.3 One Size Does Not Fit All

With all the number of OO processes available, how can we choose the most appropriate process that is the best fit for target software project? A difficult task simply because there is no one size does fit all [Constantine and Lockwood, 1994]. Fayad *et al.* have similarly stated this fact by saying that "there are no meaningful one-size-fits-all software development methods" [Fayad et al., 1996]. For example, very large projects require more planning, management and control than small projects, which, in contrast, may better utilize an agile/lightweight process. Intensive, critical and real time projects need more safety, quality assurance, reviews and accurate and extensive testing than other projects. On-line web applications require a different process to ordinary systems, as mentioned in [Lowe et al., 2001][Haire et al., 2002], for instance. Brand new systems require a different process to software re-engineering projects. Soft systems (ill-defined) require a different process to well-defined systems. A single project will require a different process to a whole suite of projects (programme). Organizations transitioning to Object Technology (OT) for the first time definitely require a different process to than other organizations that have already moved to, and adopted OT.

3.4 Process Selection Criteria

Criteria for selecting a software process mainly focus on the best match between the process and the software development environment within the problem domain that leads to economical advantages. Other criteria include:

- Alignment with the organization's strategy as this is considered by many as a Critical Success Factor for software development [Yourdon, 1999].
- Project size since large projects usually require more sophisticated and formal process such as OPEN or RUP whereas small projects generally need a lightweight and fast process such as eXtreme Programming [Beck, 1999].
- Project timeframe and other constraints that developers have to satisfy.
- Development type since web and on-line development requires a different process than business or distributed applications.
- Safety requirements since critical systems such as life support and airline navigation systems mostly emphasise safety issues.
- Other process selection criteria including scalability, complexity, architecture, reliability, maintainability, support, cost and development life span.

3.5 Process Maturity

The process maturity (or capability) framework arose from the need to rate potential software development organizations on their ability to deliver product on time, to budget, within requirements and the to required quality levels [ISO, 1998][SEI, 1993]. Usually, the framework consists of 5 levels, beginning with the lowest level of 1 and ending with the highest level of 5. At the lowest level, an organization develops software relying on "heroic" effort and seldom delivers to schedule and budget. At level 2, development is managed and controlled while at level 3 software development processes are defined and followed. Levels 4 and 5 bring in some of the more advanced process management concepts such as statistical process control and continuous improvement. There have been several studies confirming a relationship between higher maturity levels and improved software development performance [Brodman and Johnson, 1996] [Butler and Lipke, 2000][Goldenson and Herbsleb, 1995][Lawlis et al., 1995]. While it is obvious that higher capability levels are better, but they incur organizational effort and cost which may not always be required or justified by a particular project. For example, a prototype development may need to emphasize speed rather than rigorous quality in order to demonstrate proof of concept. Similarly a small team of less than 10 developers may not need the same organizational management rigour as a project requiring several hundreds of developers. In other words, a process operating at a lower capability level may be acceptable, even optimal, under such constraints.

3.6 Full Development Life Cycle Support

A good process should be able to support every activity within the development life cycle from analysis to deployment and maintenance. This doesn't only include analysis, design and building, but also the support of other areas, such as project management, testing, retrospectives and enhancements. The lifecycle commences with the identification of a business problem and ends with the termination of use of the software in that organization.

The selected process should also support some important issues, including process improvement, reusability and repeatability and should be usable at each level of process capability. It should not focus only on the current problem or project; instead, it should be flexible enough to be adapted to the problem or other development projects. Ideally, the lifecycle *model* used should be able to be varied since for some projects, a waterfall approach is adequate, for others an OO fountain model is needed and for others an XP-style lifecycle model is most appropriate [see Chapter 6 of [Firesmith and Henderson-Sellers, 2002]]. A good process should maintain consistency between development activities without violating the object-oriented paradigm [Humphrey, 1988].

3.7 Reuse and its Applicability to Quality

Reusability has the potential to underpin improvements in quality and therefore needs to be integrated into the process. Reuse is as old as programming itself; it is a software technology that remains valid [Lynex and P.J., 1998], building or assembling software systems from predefined components. Reuse must be seen as a long-term strategy to which management is committed.

It is shown that reusability needs to extend not only for a project but over the entire organization. Reuse must be supported by an organizational structure that divides the roles and responsibilities of system developers into reusable component builders and assemblers [McClure, 1997]. This is part of the team building initiative that a software development team should undergo to implement reusability for software improvement.

Recently, most software developers and their management are integrating reuse into their processes knowing the cost effectiveness, low risk and reduction in time to produce a product. Modern object-oriented processes are very often seen as enablers for reuse [Morisio, 2002].

Object oriented software engineering required large scale software reuse be recognised in a practical way. The basic concept of systematic reuse is to develop systems components of a reasonable size and reuse [McIlroy, 1969]. This ensures that when components are reused they have not only been tested before, but are tested again and again when the components are reused, thus ensuring better quality of the software (or, here, the process/methodology.)

3.8 Team Building as a Prerequisite for a Quality Process

The importance of team building as part of a quality process is also frequently ignored. These more human aspects of computing are frequently not included in software development methodologies so the project managers must utilize a (perhaps incompatible) human resource (HR) strategy.

Team building has not been largely studied in the context of software development methodologies. With the introduction of Agile Methods, the emphasis of people has become more of a concern. As Agile Methods are explained to be people-focused not process-focused [Cockburn and Highsmith, 2001], software methodologies are turning towards how to improve their teams and utilize their resources.

From experience, [Fowler, 1995] states that many Information Technology projects have failed due to the nature of teams within the IT industry. Teams have been made up of IT specialists, with no other team roles to understand the users view. Teams should consist of a diversity of individuals to aid in problem solving, decision-making and change that occurs during the course of a project. Typical profiles include individuals who could be described [Belbin, 1981] as co-ordinators (respected by the team, chair the team but not necessary creative), and plants (clever people, who are innovative and have problem solving skills.) Team building involves understanding the skills and talents of the people who are required to fulfil the goals of the software process and, most importantly, how these skills complement each other.

The role of each person within a team is affected by the size of a team. A team of 4-6 people has specific roles [Belbin, 2000]. The term team, having come from team sports, illustrates that in every sport there are teams of a set size. The size allows each player to have a role to play within the team. On the other hand, [Belbin, 2000] also mentions that there is a difference between the team activity of a team sport and a team in an organization. The sporting team is trained with skills to compete on a field of play with little communication. In an organizational team, communication is vital for the decision making processes in a competitive economy that is changing rapidly each day.

Selecting a team needs careful planning and basic knowledge of the individuals required for the team [Belbin, 2000]. The roles that are selected through team building exercises are the core of the team. As each role is played out during the development of the software, the team becomes committed and motivated to the project. In our modern organizations where change is constant, our teams need to be flexible to embrace changes. With team roles, this flexibility allows the entire team to focus and remain with the organizational vision.

Building a team relies on the people resources to accomplish the team's goals in light of the organizational mission. Team building is a critical component of any high quality process, as exemplified in contemporary agile methodologies.

4 Solution: Method Engineering

Underpinning all these problems is the lack of use of a flexible software development methodology that takes into account all the human and organizational cultural issues as well as technical issues. The solution we advocate is commonly referred to as situational method engineering, or SME for short [Kumar and Welke, 1992][Brinkkemper, 1996] [Hofstede et al., 1997][Kraiem et al., 2000][Ralyté and Rolland, 2001]. This is an approach with a sound theoretical basis but, as yet, little adoption by industry. Rupprecht *et al.* have proposed that a framework together with techniques may be useful for constructing situational engineering processes (which they call "process individualization") by applying a number of predefined operators on a number of predefined generic process building blocks with the help of a reference model [Rupprecht et al., 2000]. While this approach provides a complete solution to the situational methodology problem, it is only implemented in a number of engineering domains, which are simpler and less volatile than the field of software engineering. Their proposals included a CASE tool for process construction, commercially available as the ARIS Process Generator. Further development of such tools, more appropriately named CAME (Computer Assisted Method Engineering), is given by [Saeki, 2003].

In the SME approach, methodological ideas are arranged as "method chunks" or "process components" and stored in a common repository [Ralyté and Rolland, 2001] [Firesmith and Henderson-Sellers, 2002]. A number of these are selected for process construction and customization, selection being determined by a large number of factors highly specific to the particular software development organization or particular project. Rules for constructing such processes are still being investigated [Klooster et al., 1997] [Brinkkemper et al., 1999][Henderson-Sellers, 2002][Ralyté, 2002].

A process for method assembly was proposed by [Ralyté and Rolland, 2001], which consists of two parts: the Method Engineering Process Model (MEPM) and the Assembly Process Model (APM). The authors identify three different cases that need to be addressed by the processes, namely: in the definition of a brand new methodology created to satisfy a set of methodology requirements; for adding alternative method chunks to an existing methodology; to add new functionality to an existing methodology. Each of these three cases is represented as an individual strategy that is used to link the requirements to the constructed method chunk. The process is described graphically in terms of a map between one node representing "Specify Method Requirements" and a second node named "Construct a Method Chunk". They also include a "reverse" strategy from "Construct a Method Chunk" back to "Specify Method Requirements", a strategy called "Requirements Correction".

Based on this high level process construction process of the MEPM, the finer detail is added using the APM. The APM focuses on a node called "Select a Chunk" which has four associated strategies (evaluation, decomposition, aggregation and refinement). These strategies are used to evaluate the appropriateness of each selected chunk, checking whether it meets the methodological requirements.

1. There should be at least one concept newly introduced in each method fragment.
2. There should be at least one concept linking the two fragments to be assembled.
3. When adding new concepts, there should be connections between them and existing fragments.
4. When adding new associations, both new fragments should be participants.
5. In the resultant combined fragment, there should be no isolated elements.
6. There should be no name duplication for different method fragments.
7. Identification of added concepts should occur after the associated concepts have been identified.
8. When two fragments are assembled, it is necessary that the output of one is used as the input to the other.
9. Every work product must be identifiable as the output of a particular process fragment.
10. When a work product has been created from other work products, then the process fragments producing the individual work products are summed to the process producing the amalgamated work product.
11. Any technical method fragment should be supported by a conceptual method fragment.
12. When there is an association between two product fragments, there should be at least one association between their respective components.

Table 3: Rules proposed in [Brinkkemper et al., 1999] for ensuring constructed process quality

Once the chunk is selected correctly, then it is likely that this chunk will need to be incorporated with other chunks. There are two basic strategies proposed: an association strategy when there is no overlap between the new chunk and any existing chunks and, secondly, an integration strategy when there is overlap present. Integration clearly provides the largest challenges. Detailed strategies advocated for use in APM include name unification, merge, transformation, generalization, specialization, remove, addition and completeness [Ralyté and Rolland, 2001].

It is important that the constructed methodology is meaningful. This is addressed by [Brinkkemper et al., 1999], who propose a list of 12 rules [see Table 3] to ensure that the constructed methodology is of quality. It is suggested that method fragments should be viewed as being on one of three dimensions: perspective, abstraction level and granularity. Violations of integrity are classified in [Brinkkemper et al., 1999] as either:

Internally incomplete: if a fragment requires another, e.g., in providing a required work product.

Inconsistent: if a conflict between two fragments arise.

Inapplicable: when the use of a fragment is not possible due to lack of capability (important in the context of this paper.)

In this approach, the rules for method assembly are represented using a formal logic. This is used to define all small scale manipulations as well as the completeness and consistency rules.

An important component of method engineering is to ensure that the method chunks or process components are congruent with (and instantiated from) a process metamodel [Brinkkemper et al., 1999][Henderson-Sellers, 2003]. Several such metamodels have been proposed, including an ER-based one [Kelly et al., 1996], object-oriented metamodels [Firesmith and Henderson-Sellers, 2002][OMG, 2003], or even so-called virtual workspaces and capability assessment [Hawryszkiewycz, 2000]. More general discussion of this topic is also found in [Brinkkemper et al., 1999], for instance.

Finally, transitioning an organization to a new technology (here method engineering) requires consideration of both personal and organizational cultures. Furthermore, team building [see Section 3.8] will diversify process ownership since adopting a technology is a team responsibility, not an individual one. It has also been found from observations that, in practice, this introduction is best done in an incremental fashion. The ability to incrementally introduce a method also leads to support for software process improvement and the much needed agility of a process to fit both current and future situations.

A final thread is the desirability to automate the process/method construction process; appropriate supporting tools are being developed, but there are not conclusive results [Nguyen and Henderson-Sellers, 2003b][Nguyen and Henderson-Sellers, 2003a].

Industrial evaluations of the approach using the OPF [Serour et al., 2002] demonstrate the viability of an SME approach to software development. These evaluative industry studies were conducted with a legal publishing company in Sydney, Australia, who were transitioning to an object-oriented, web-based development environment. This transition was facilitated by the adoption by the organization of a planned transition process and the use of a method engineering approach by which they were able to adopt, initially, an agile version of OPEN (with UML as the notation) and then allow it to be incrementally improved by the addition of increasingly sophisticated process components, compatible with higher levels of process capability [see Section 3.5].

4.1 A Practical Example of Method Engineering

To illustrate how such a methodology might look in practice, examples have been taken from a methodology that was developed as part of the OOSPICE project. The two examples show the same process, namely: a basic process listing only those tasks necessary to achieve a SPICE capability level of 1 [see Section 4.1.1], and listing all tasks necessary to achieve a level 5, too [see Section 4.1.2]. The examples, generated by a tool embodying many of the integrity constraints discussed earlier, show how any process described in this manner can easily be tailored to the capability level required by the organization or by a specific project.

In [Section 4.1.2], the capability level is shown on the left hand side for each entry under *Outcomes*, *Tasks* and *Techniques*. The process can be tailored still further by selecting techniques (only a few are shown for illustrative purposes) with which to complete specific tasks that are appropriate to process capability level required by the organization or a specific project.

4.1.1 Example Process at Capability Level 1

Purpose: To establish a managed and evolving repository of knowledge about the domain.

Outline: A strategy to guide the management of the repository of domain knowledge is developed. The scope and characteristics of the domain are defined and standard terminology is established. Finally, the domain status is assessed.

Outcomes:

1	A domain engineering strategy is developed.
1	The domain is defined.
1	Standard terminology is established.
1	Domain characteristics are identified.
1	The domain status is assessed.

Specific Tasks:

1 **Develop a strategy for domain engineering**

Identify the stakeholders and their objectives, organizational constraints and applicable organizational policies. Identify how to achieve the desired outcomes while satisfying the stakeholders' objectives within the identified constraints. Identify what needs to be done and when, the required resources and how to overcome deficits in those resources. Identify how the organization will decide the strategy is working.

Actions:

Domain Strategy	Create
Business Objectives	ReadOnly

Techniques:

Strategy Development	Optional

1 **Define domain**

A description of the type of problem that systems in the domain solve and the external environment with which systems interact is produced. The observable behaviour that systems exhibit in solving the problem is described. The behavioural aspects of the system should be viewed from a black-box perspective. Any significant constraints concerning how the systems operate in terms of performance, reliability or distribution concerns are also included. The primary functions performed by every system in the domain and any important functions performed by only some systems are defined.

Actions:

Domain Specification	Create
Domain Strategy	ReadOnly

1 **Establish standard terminology**

Definitions of all significant terms used by domain experts in discussing the requirements or engineering of systems in the domain. Cross references to related terms and references to definitions from standard glossaries should be included. A structure should be created that shows term specializations and relationships among similar concepts to reveal missing terms that represent generalizations of specializations of known terms. In addition, a structure should be created that shows the composition of terms and the interrelationship of independent concepts in the formation of logical structures to reveal missing terms that are necessary to complete the definition of other terms or terms that tie other terms together into more complex concepts.

Actions:

Domain Specification	Modify

1 **Identify domain characteristics**

Assumptions, constraints and core characteristics of the domain are identified.

Actions:

Domain Specification	Modify

1 **Assess domain status**

An evaluation of the technical maturity of the domain in terms of Domain Objectives and plans for domain development and evolution is performed. The Assess Domain Status task is defined in terms of its sub-activities, namely: (i) Determine Marketability, and (ii) Determine Implement Ability and Risk. These sub-activities comprise a Viability Analysis. Each of the sub-activities results in an endorsement and commitment to the defined Domain Scope.

Actions:

Domain Assessment	Create
Domain Specification	ReadOnly
Domain Strategy	ReadOnly

4.1.2 Example Process at Capability Level 5

Purpose: To establish a managed and evolving repository of knowledge about the domain.

Outline: A strategy to guide the management of the repository of domain knowledge is developed. The scope and characteristics of the domain are defined and standard terminology is established. Finally, the domain status is assessed.

Outcomes:

1 A domain engineering strategy is developed.
1 The domain is defined.
1 Standard terminology is established.
1 Domain characteristics are identified.
1 The domain status is assessed.

2 The objectives for the performance of the process are identified. (2.1)
2 Resources required for performing the process are made available, allocated and used. (2.1)
2 The responsibility and authority for performing the process activities is assigned. (2.1)
2 The performance of the process is planned.(2.1)
2 The performance of the process is controlled. (2.1)
2 The process work product requirements are defined. (2.2)
2 The documentation and control requirements for the process work products are defined. (2.2)
2 Any dependencies among controlled work products are identified. (2.2)
2 Work products are appropriately identified, documented and controlled. (2.2)
2 Work products are verified and, if necessary, adjusted to meet the defined requirements. (2.2)

3	A standard process including appropriate guidance on tailoring is defined, that supports the execution of the process. (3.1)
3	Performance of the defined process is conducted in accordance with an appropriately selected and/or tailored standard process. (3.1)
3	Process performance data is collected and used as a basis for understanding the behaviour of the defined process. (3.1)
3	Experiences of using the defined process are collected and used to refine the standard process and tailoring guidance. (3.1)
3	The standard process identifies the competencies, roles and responsibilities required for enacting the defined process. (3.2)
3	The process infrastructure required for performing the defined process is identified and documented as part of the standard process. (3.2)
3	The required resources are made available, allocated and used to support the performance of the defined process. (3.2)
4	Objectives for process performance are established. (4.1)
4	Product and process measures are identified in line with relevant process objectives. (4.1)
4	Product and process measures are collected to monitor the extent to which the defined process objectives are met. (4.1)
4	Process capability is measured and maintained across the organizational unit. (4.1)
4	Suitable analysis and control techniques are identified. (4.2)
4	During process enactment, product and process measures are analysed to support control of process performance within defined limits. (4.2)
4	Process performance trends across the organizational unit are analyzed. (4.2)
4	Effective actions are taken to address special causes of variation in performance. (4.2)
5	The process improvement goals for the process are defined that support the relevant business goals of the organization. (5.1)
5	The causes of real and potential variations are identified. (5.1)
5	Improvement opportunities are identified. (5.1)
5	An implementation strategy is established and deployed to achieve the process improvement goals across the organization. (5.1)
5	The impact of all proposed changes is assessed against the objectives of the defined process and standard process. (5.2)
5	The implementation of all agreed changes is managed to ensure that any disruption to the process performance is understood and acted upon. (5.2)
5	The effectiveness of process change on the basis of actual performance is evaluated against the defined product requirements and process objectives to determine whether results are due to common or special causes. (5.2)

Specific Tasks:

1 **Develop a strategy for domain engineering**

Identify the stakeholders and their objectives, organizational constraints and applicable organizational policies. Identify how to achieve the desired outcomes while satisfying the stakeholders' objectives within the identified constraints. Identify what needs to be done and when, the required resources and how to overcome deficits in those resources. Identify how the organization will decide the strategy is working.

<u>Actions:</u>

Domain Strategy	Create
Business Objectives	ReadOnly

<u>Techniques</u>

Strategy Development	Optional

1 **Define domain**

A description of the type of problem that systems in the domain solve and the external environment with which systems interact is produced. The observable behaviour that systems exhibit in solving the problem is described. The behavioural aspects of the system should be viewed from a black-box perspective. Any significant constraints concerning how the systems operate in terms of performance, reliability or distribution concerns are also included. The primary functions performed by every system in the domain and any important functions performed by only some systems are defined.

Actions:

Domain Specification	Create
Domain Strategy	ReadOnly

1 **Establish standard terminology**

Definitions of all significant terms used by domain experts in discussing the requirements or engineering of systems in the domain. Cross references to related terms and references to definitions from standard glossaries should be included. A structure should be created that shows term specializations and relationships among similar concepts to reveal missing terms that represent generalizations of specializations of known terms. In addition, a structure should be created that shows the composition of terms and the interrelationship of independent concepts in the formation of logical structures to reveal missing terms that are necessary to complete the definition of other terms or terms that tie other terms together into more complex concepts.

Actions:

Domain Specification	Modify

1 **Identify domain characteristics**

Assumptions, constraints and core characteristics of the domain are identified.

Actions:

Domain Specification	Modify

1 **Assess domain status**

An evaluation of the technical maturity of the domain in terms of Domain Objectives and plans for domain development and evolution is performed. The Assess Domain Status task is defined in terms of its sub-activities: (i) Determine Marketability, and (ii) Determine Implement Ability and Risk. These sub-activities comprise a Viability Analysis. Each of the sub-activities results in an endorsement and commitment to the defined Domain Scope.

Actions:

Domain Assessment	Create
Domain Specification	ReadOnly
Domain Strategy	ReadOnly

2 **Develop a plan for the process**

Identify the process objectives, input and output work products, resources required to achieve the process objectives and the resources necessary to achieve them. Identify responsibilities and authorities necessary for performing the process.

Actions:

Process Plan	Create
Business Objectives	ReadOnly
Project Plan	ReadOnly

Techniques:

Timeboxing	Optional
Workflow Analysis	Optional

2 **Identify work product control requirements**

For all process work product, identify requirements related to their control and management through the product's life cycle.

Actions:

Process Plan	ReadOnly
Work Product Requirements	ReadOnly

2 **Identify work product dependencies**

Identify and document dependencies among work products.

Actions:

Process Plan	Modify
Process Work Product	Modify

2 **Identify work product requirements**

For each process work product, identify its content and project requirements.

Actions:

Work Product Requirements	Create
Process Plan	ReadOnly

2 **Manage work products**

Identify, document and control work products. Work products should be appropriately identified by name, status and version. Changes to work products should be managed and updated versions of work products communicated to stakeholders.

Actions:

Process Work Product	Modify

2 **Modify controlled work product**

A controlled work product is modified and the modifications recorded. Updated versions are placed under control and distributed as necessary.

Actions:

Process Work Product	Modify

2 **Monitor and manage the process plan**

Monitor a planned process according to its plan. Detect deviations from the plan are detected and resolved. Issues arising from performing the process are communicated to stakeholders.

Actions:

Process Plan	Modify

2 **Verify work products**

Work products are verified against their requirements.

Actions:

Work Product Requirements	ReadOnly

3 **Collect process performance data**

Collect and analyse process performance data. Use the analysed data to contribute to understanding and improving the standard process.

Actions:

Defined Process	ReadOnly
Process Plan	ReadOnly

3 **Define a standard process**

A standard process that supports the execution of the process is defined. Tailoring guidelines and constraints on such tailoring are written to describe how the standard process could be modified to suit the particular project or circumstances. The definition should cover roles, responsibilities and infrastructure necessary to perform the process.

Actions:

Defined Process	Create

3 **Improve the standard process**

Data from and experiences with performed processes are used to develop improvements to the standard processes.

Actions:

Defined Process	Modify

3 **Perform defined process**

The defined, and possibly tailored, process is performed as defined and planned. All required work products will be produced.

Actions:

Defined Process	ReadOnly

3 **Support process performance**

Required resources are made available, allocated and used to support the performance of the process. Another critical aspect of this process attribute is ensuring that enabling conditions for successful deployment (implementation) of the defined process are present. Enabling conditions include: (i) defining the specific attributes of human resources who implement the process, (ii) understanding the process infrastructure required for performing the defined process, (iii) successful allocation and deployment of the required human resources and process infrastructures, (iv) a common documented understanding of roles, responsibilities and competencies for performing the defined process.

The process infrastructure encompasses tools, methods and special facilities that are required for performing the defined process.

Actions:

Defined Process	ReadOnly
Process Plan	ReadOnly

4 **Analyse process performance data**

Industry best practice and process performance data are analysed to identify potential opportunities for process improvement.

Actions:

Process Performance Analysis	Create
Process Performance Data	ReadOnly

4 **Analyse process performance trends**

Analyse process performance data from across the organizational unit to determine causes of variation of process performance and the extent to which the process supports the business objectives.

Actions:

Process Performance Analysis	Create
Process Measurement Plan	ReadOnly
Process Performance Data	ReadOnly

4 **Assess process capabiltity**

Measure and record process capability across the organizational unit.

Actions:

Process Capability Assessment	Create
Defined Process	ReadOnly

4 **Control the process performance**

Use statistical process control or other suitable quantitative technique to monitor and control the process. The analysis and control techniques chosen will be influenced by the nature of the process as well as by the overall context of the organizational unit being assessed. For example, not all processes are equally suited to statistical control, and alternative techniques can be selected that demonstrate a qualitative understanding of the process.

Actions:

Process Measurement Plan	ReadOnly
Process Performance Data	ReadOnly

4 **Correct process performance**

Actions are identified to correct causes of process performance variation and to better achieve the process objectives.

Actions:

Defined Process	Modify
Process Performance Analysis	ReadOnly

4 **Define process analysis and control techniques**

Define process analysis and control techniques that are capable of identifying root causes of variation in process performance.

Actions:

Process Measurement Plan	Modify

4 **Define process performance measures**

Measures that support the process objectives are identified and documented.

Actions:

Process Measurement Plan	Create

4 **Define process performance objectives**

Relevant business goals are understood and clearly identified, and some form of correspondence is established between the business goals and the specific goals and measures for product and process.

Actions:

Process Measurement Plan	Modify

4 **Quantitatively monitor process performance**

Data for the identified process measures is collected and analysed to determine the extent to which the defined process supports the objectives for that process.

Actions:

Defined Process	ReadOnly
Process Measurement Plan	ReadOnly
Process Performance Data	ReadOnly

5 **Assess proposed process changes**

Assess the potential impact of proposed process changes against the process objectives.

Actions:

Process Improvement Plan	ReadOnly
Process Improvements	ReadOnly

5 **Define process improvement goals**

Define process improvement goals that support business goals.

Actions:

Process Improvement Plan	Create

5 **Develop and implement improvement strategy**

Develop a strategy to implement process improvements. Implement the strategy across the organization to achieve the identified improvements.

Actions:

Process Improvement Plan	Modify

5 **Develop process improvement strategy**

The implementation timing and sequencing of agreed changes is carefully planned so as to ensure a minimal amount of disruption to process performance. This planning will typically consider factors such as project criticality and status, process change effectiveness evaluation and new business generation.

Actions:

Process Improvement Plan	ReadOnly
Project Plan	ReadOnly

5 **Evaluate process change effects**

The effectiveness of changes is evaluated against actual results and adjustments are made as necessary to achieve relevant process improvement objectives.

Actions:

Defined Process	ReadOnly
Process Improvement Plan	ReadOnly
Process Improvements	ReadOnly

5 **Identify process improvements**

Identify potential process improvements.

Actions:

Process Improvements	Create
Process Improvement Plan	ReadOnly

5 **Identify real and potential task variation causes**

Examine processes to anticipate and identify real or potential causes of variations in process.

Actions:

Business Objectives	ReadOnly
Defined Process	ReadOnly
Process Performance Data	ReadOnly

5 **Manage process change implementation**

The implementation timing and sequencing of agreed changes is carefully planned so as to ensure a minimal amount of disruption to process performance. This planning will typically consider factors such as project criticality and status, process change effectiveness evaluation and new business generation.

Actions:

Defined Process	Modify
Process Improvement Plan	ReadOnly
Process Improvements	ReadOnly

4.2 Improving Quality with Method Engineering

Improvements to the twin goals of software productivity and quality have been sought from the earliest days of commercial software development. The question to be examined here is not whether this approach is better than nothing, but whether it is better than other methodologies.

The most (and most rigorous) information on software productivity and quality improvements as a consequence of adopting a particular software development methodology comes from studies associated with the Software Engineering Institute's Capability Maturity Model (CMM) and its successor the Integrated Capability Maturity Model (CMMI). The CMM grew out of work at the Software Engineering Institute at Carnegie Mellon University on a U.S. Air Force project. The objective was to provide guidance to the military services in selecting capable software contractors [Humphrey, 1989]. The subsequent CMM was adopted by many military-related organizations. There have been several reports of the effects that adopting the CMM has on software quality and productivity, among other things. [Herbsleb et al., 1994] related improvements with the number of years that organizations had spent engaged in software process improvements rather than on the maturity level attained. The report concluded that the return on investment when "improvement is planned and executed well and takes place in a favorable environment" was five or six times the cost of implementing the improvements. [Goldenson and Herbsleb, 1995] found that maturity level was positively related to organizational performance as measured by its ability to meet schedules, ability to meet budget, product quality, staff productivity, customer satisfaction and staff morale. [Lawlis et al., 1995] examined the relationship between maturity level and cost performance and maturity level and schedule performance but not between maturity level and product quality. They found that maturity level was positively related to both.

The successor to the CMM, the CMMI, shows similar relationship to improved product quality, reduced development cost and reduced development schedule but there is no analysis relating the reported benefits to the maturity level attained by the organizations that participated in the study [Goldenson and Gibson, 2003].

From these studies, it can be concluded that a capability based methodology allows organizations to attain a level of maturity, and hence of software development performance, that suits their circumstances and that the maturity level chosen or attained will affect their productivity and quality performance.

CMM and CMMI are not the only capability level based methodologies. Although SPICE [ISO, 1998] is a software process assessment method rather than a software development methodology, it does contain an exemplar process model that serves as a software development methodology. To date, relationships between assessed SPICE capability levels and software development quality and productivity have been anecdotal rather than quantitative, and have concentrated on process improvement rather than specific outcomes related to using the SPICE assessment framework or its contained software development methodology. Certainly the assumption behind many papers is that process improvement sufficiently achieves its objectives to make process improvement attractive. To date there has been no complete field trial of the described methodology.

5 Conclusions

Situational method engineering and its industrial testing provides an approach to software engineering that will permit organizations to enhance their success rate. While no silver bullet, SME creates an environment in which a software development organization can construct both a highly customized process for its current situation, but also one that will grow in sophistication. Consequently, the organization can increase its capability, as measured by the CMM or SPICE scales. Metrics-focused method chunks can be incorporated into the constructed organizationally-specific methodology increasingly as this transition along the capability scale occurs.

Acknowledgements

C.G-P. and M.K.S. wish to acknowledge research funding support from the Australian Research Council. This is Contribution number 03/21 of the Centre for Object Technology Applications and Research.

References

[Abrahamsson et al., 2003] Abrahamsson, P., Warsta, J., Siponen, M., and Ronkai nen, J. (2003). New directions on agile methods a comparative analysis. In *Proc. of ICSE'03*, pages 244–254. IEEE Computer Society Press.

[Baddoo and Hall, 2003] Baddoo, N. and Hall, T. (2003). De-motivators for software process improvement: an analysis of practitioners´ views. *Journal of Systems and Software*, 66:23–33.

[Basili and Rombach, 1987] Basili, V. and Rombach, H. (1987). Tailoring the software process to project goals and environments. In *Proc. of ICSE'98*, pages 345–357. IEEE Computer Society Press.

[Bauch and Chung, 2001] Bauch, G. and Chung, C. (2001). A statistical project control tool for engineering managers. *Project Management Journal*, 32:37–44.

[Beck, 1999] Beck, K. (1999). *Extreme Programming Explained. Embrace Change*. Addison-Wesley.

[Beedle et al., 2002] Beedle, M., Devos, M., Sharon, Y., Schwaber, K., and Sutherland, J. (2002). SCRUM: An extension pattern language for hyper productive software development. `http://www.tiac.net/users/jsuth/scrum`.

[Belbin, 2000] Belbin, M. (2000). *Beyond the Team.* Butterworth-Heinemann.

[Belbin, 1981] Belbin, R. (1981). *Management Teams: Why They Succeed or Fail.* Butterworth Heinemann.

[Boehm et al., 2000] Boehm, B., Abts, C., Brown, A., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D., and Steece, B. (2000). *Software Cost Estimation with Cocomo II*. Prentice Hall.

[Breton and Bézivin, 2001] Breton, E. and Bézivin, J. (2001). Model driven process engineering. In *Proc. of the 25th Annual International Computer Software and Applications Conference*, pages 225–230. IEEE Computer Society Press.

[Brinkkemper, 1996] Brinkkemper, S. (1996). Method engineering: engineering of information systems development methods and tools. *Information Software Technology*, 38(4):275–280.

[Brinkkemper et al., 1999] Brinkkemper, S., Saeki, M., and Harmsen, F. (1999). Meta-modelling based assembly techniques for situational method engineering. *Information Systems*, 24(3):209–228.

[Brinkkemper et al., 2001] Brinkkemper, S., Saeki, M., and Harmsen, F. (2001). A method engineering language for the description of systems development methods (extended abstract). In Dittrich, K., Geppert, A., and Norrie, M., editors, *Advanced Information Systems Engineering*, number 2068 in LNCS, pages 473–476. Springer-Verlag.

[Brodman and Johnson, 1996] Brodman, J. and Johnson, D. (1996). Return on investment from software process improvement as measured by u.s. industry. *Crosstalk*, 9.

[Butler and Lipke, 2000] Butler, K. and Lipke, W. (2000). Software process achievement at tinker air force base, oklahoma. Technical report, Software Engineering Institute.

[Chroust, 2000] Chroust, G. (2000). Software process models: structure and challenges. In Feng, Y., Notkin, D., and Gaudel, M., editors, *Proc. of the IFIP Congress 2000*, pages 279–286.

[Cockburn, 2000] Cockburn, A. (2000). Selecting a project's methodology. *IEEE Software*, 17(4):64–71.

[Cockburn, 2001] Cockburn, A. (2001). *Agile Software Development*. Addison Wesley.

[Cockburn and Highsmith, 2001] Cockburn, A. and Highsmith, J. (2001). Agile software development: The people factor. *IEEE Computer*, 34(11):131–133.

[Constantine, 2002] Constantine, L. (2002). Non-skid agility? *Information Age*, August/September:34–42.

[Constantine and Lockwood, 1994] Constantine, L. and Lockwood, L. (1994). One size does not fit all: fitting practices to people. *American Programmer*, 7(12):30–38.

[Cunin et al., 2001] Cunin, P.-Y., Greenwood, R., Francou, L., Robertson, I., and Warboys, B. (2001). The pie methodology: Concept and application. In Ambriola, V., editor, *Proc. of the 8th European Workshop*, number 2077 in LNCS, pages 3–26. Springer Verlag.

[D'Souza and Wills, 1999] D'Souza, F. and Wills, A. (1999). *Objects, Components and Frameworks with UML: The Catalysis Approach*. Addison-Wesley.

[Fayad et al., 1996] Fayad, M., Tsai, W., and Fulghum, M. (1996). Transition to object-oriented software development. *Communications of the ACM*, 39(2):108–121.

[Firesmith and Henderson-Sellers, 2002] Firesmith, D. and Henderson-Sellers, B. (2002). *The OPEN Process Framework: An Introduction*. Addison-Wesley.

[Fowler, 1995] Fowler, A. (1995). How to build effective teams. *People Management*, 1(4).

[Fowler, 2002] Fowler, M. (2002). The new methodology. `http://www.martinfowler.com/articles/newMethodology.html`.

[Gasson, 1998] Gasson, S. (1998). Framing design: a social process view of information system development. In *Proc. of the International Conference on Information Systems*.

[Glass, 2000a] Glass, R. (2000a). Process diversity and a computing old wives'/husbands' tale. *IEEE Software*, 17(4):128–127.

[Glass, 2000b] Glass, R. (2000b). Talk about a software crisis: Not! *Journal of Systems and Software*, 55:1–2.

[Goldenson and Gibson, 2003] Goldenson, D. and Gibson, D. (2003). Demonstrating the impact and benefits of cmmi: An update and preliminary results. Technical report, Software Engineering Institute.

[Goldenson and Herbsleb, 1995] Goldenson, D. and Herbsleb, J. (1995). After the appraisal: A systematic survey of process improvement, its benefits, and factors that influence success. Technical report, Software Engineering Institute.

[Graham et al., 1997] Graham, I., Henderson-Sellers, B., and Younessi, H. (1997). *The OPEN Process Specification*. Addison Wesley.

[Group, 1994] Group, T. S. (1994). The chaos report. `http://www.pm2go.com/sample_research/chaos_1994_1.php`.

[Gruhn, 1992] Gruhn, V. (1992). Software processes are social processes. In *Proc. of Fifth International Workshop on Computer-Aided Software Engineering*.

[Haire et al., 2002] Haire, B., Lowe, D., and Henderson-Sellers, B. (2002). Supporting web development in the OPEN process: additional roles and techniques. In Bellahsène, Z., Patel, D., and Rolland, C., editors, *Object-Oriented Information Systems*, number 2425 in LNCS, pages 82–94. Springer Verlag.

[Hawryszkiewycz, 2000] Hawryszkiewycz, I. (2000). Knowledge networks in administrative systems. In *Working Conference on Advances in Electronic Government*, pages 59–75.

[Henderson-Sellers, 2002] Henderson-Sellers, B. (2002). Process metamodelling and process construction: examples using the open process framework (OPF). *Annals of Software Engineering*, 14:341–362.

[Henderson-Sellers, 2003] Henderson-Sellers, B. (2003). Method engineering for oo system development. *Communications of the ACM*, 46(10):73–78.

[Henderson-Sellers et al., 2002] Henderson-Sellers, B., Bohling, J., and Rout, T. (2002). Creating the oospice model architecture: A case of reuse. In Rout, T., editor, *Proc. of SPICE 2002*, pages 171–181.

[Henderson-Sellers et al., 1998] Henderson-Sellers, B., Simons, A., and Younessi, H. (1998). *The OPEN Toolbox of Techniques*. Addison-Wesley.

[Henninger et al., 2002] Henninger, S., Ivaturi, A., Nuli, K., and Thirunavukkaras, A. (2002). Supporting adaptable methodologies to meet evolving project needs. In *Proc. of the 1st ICSE'02 Workshop on Iterative, Adaptive, and Agile Processes*.

[Herbsleb et al., 1994] Herbsleb, J., Carleton, A., Rozum, J., Siegel, J., and Zubrow, D. (1994). Benefits of cmm-based software process improvement: Initial results. Technical report, Software Engineering Institute.

[Hofstede et al., 1997] Hofstede, T., A.H.M., and Verhoef, T. (1997). On the feasibility of situational method engineering. *Information Systems*, 22:401–422.

[Hruby, 2000] Hruby, P. (2000). Designing customizable methodologies. *Journal on Object-Oriented Programming*, 13(8):22–31.

[Hughes and Cotterell, 1999] Hughes, B. and Cotterell, M. (1999). *Software Project Management*. McGraw-Hill.

[Humphrey, 1988] Humphrey, W. (1988). Characterizing the software process. *IEEE Software*, 5(2):73–79.

[Humphrey, 1989] Humphrey, W. (1989). *Managing the Software Process*. Addison-Wesley.

[Humphrey, 1994] Humphrey, W. (1994). *A Discipline for Software Engineering*. Addison-Wesley.

[Humphrey, 2000] Humphrey, W. (2000). The team software process. Technical report, Software Engineering Institute.

[Hunter, 1998] Hunter, R. (1998). Ad project portfolio management. In *Proc. of the Gartner Group IT98 Symposium*. CD-ROM.

[ISBSG, 2003] ISBSG (2003). International software benchmarking standards group web site. `http://www.isbsg.org.au`.

[ISO, 1998] ISO (1998). Iso/iec tr 15504 — information technology: software process assessment. Technical report, International Organization for Standardization.

[ISO, 1999] ISO (1999). Iso/iec 16326:1999 — software engineering: Guide for the application of iso/iec 12207 to project management. Technical report, International Organization for Standardization.

[ISO, 2002] ISO (2002). Iso/iec software life cycle processes, amendment 1, iso/iec 12207. Technical report, International Organization for Standardization.

[Jalote, 2002] Jalote, P. (2002). Lessons learned in framework-based software process improvement. In *Proc. of the 9th Asia-Pacific Conference on Software Engineering*, pages 261–268. IEEE Computer Society Press.

[Johnson et al., 2001] Johnson, J., Boucher, K. D., Conners, K., and Robinson, J. (2001). Collaborating on project success. http://www.softwaremag.com/archive/2001feb/CollaborativeMgt.html.

[Kelly et al., 1996] Kelly, S., Lyytinen.K., and Rossi, M. (1996). Metaedit+: A fully configurable multi-user and multi-tool case and came environment. In Constapoulos, P., Mylopoulos, J., and Vassiliou, Y., editors, *Proc. of the CAISE'96 Conference*, pages 1–21. Springer-Verlag.

[Klooster et al., 1997] Klooster, M., Brinkkemper, S., Harmsen, F., and Wijers, G. (1997). Intranet facilitated knowledge management: A theory and tool for defining situational methods. In *Proc. of CAISE'97*, pages 303–317. Springer Verlag.

[Kraiem et al., 2000] Kraiem, N., Bourguiba, I., and Selmi, S. (2000). Situational method for information system project. http://www.ssgrr.it/en/ssgrr2000/papers/283.pdf.

[Kruchten, 1999] Kruchten, P. (1999). *The Rational Unified Process, An Introduction*. Addison-Wesley.

[Kumar and Welke, 1992] Kumar, K. and Welke, R. (1992). Methodology engineering: a proposal for situation-specific methodology construction. In Cotterman, W. and Senn, J., editors, *Challenges and Strategies for Research in Systems Development*, pages 257–269. John Wiley.

[Laplante, 1998] Laplante, A. (1998). Border war. *Computer World*, 9(March):81–84.

[Lawlis et al., 1995] Lawlis, D., Flowe, C., and Thordahl, C. (1995). A correlational study of the cmm and software development performance. *Crosstalk*, 8.

[Lowe et al., 2001] Lowe, D., , and Henderson-Sellers, B. (2001). Characteristics of web development processes. In Milutinovic, V., editor, *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*.

[Lynex and P.J., 1998] Lynex, A. and P.J., L. (1998). Organizational considerations for software reuse. *Annals of Software Engineering*, 5:105–124.

[MacIsaac, 2003] MacIsaac, B. (2003). An overview of the rup as a process engineering platform: Process engineering for object-oriented and component-based development. In *Proc. of the OOPSLA 2003 Workshop*, pages 43–52.

[McClure, 1997] McClure, C. (1997). *Software Reuse Techniques*. Prentice Hall.

[McConnell, 1998] McConnell, S. (1998). *Software Project Survival Guide*. Microsoft Press.

[McIlroy, 1969] McIlroy, D. (1969). Mass produced software components. In *Proc. of the 1968 NATO conference on Software Engineering*, pages 138–155.

[Morisio, 2002] Morisio, M. (2002). Success and failure factors in software reuse. *IEEE Transactions on Software Engineering*, 28(4):340–357.

[Muller, 1982] Muller, F. (1982). Definition of construction management. In *Specialty Conference on Engineering and Construction Projects*.

[Nguyen and Henderson-Sellers, 2003a] Nguyen, V. and Henderson-Sellers, B., editors (2003a). *OPENPC: a tool to automate aspects of method engineering*.

[Nguyen and Henderson-Sellers, 2003b] Nguyen, V. and Henderson-Sellers, B. (2003b). Towards automated support for method engineering with the open approach. In *Proc. of 7th IASTED International Conference on Software Engineering and Applications*, pages 691–696. ACTA Press.

[OMG, 2003] OMG (2003). Software process engineering metamodel specification, version 1.0.

[Onoma, 1987] Onoma, A. (1987). Solving the software crisis: toward management of large scale software development. In *Proc. of 1987 Fall Computer Conference on Exploring technology*, pages 244–245.

[Patton, 2003a] Patton, J. (2003a). Hitting the target: adding interaction design to agile software development. In *Proc. of the OOPSLA'02 Conference*. ACM Press.

[Patton, 2003b] Patton, J. (2003b). Improving agility: adding usage-centered design to agile software development, performance by design. In Constantine, L., editor, *Proc. of forUSE 2003 Second International Conference on Usage-Centered Design*, pages 59–73. Ampersand Press.

[Paulk et al., 1993] Paulk, C., Weber, C., Garcia, S., Chrissis, M., and Bush, M. (1993). Key practices of the capability maturity model, version 1.1. Technical report, Software Engineering Institute.

[Pérez et al., 1995] Pérez, G., El Amam, K., and Madhavji, N. (1995). Customising software process models. In *Proc. of the 4th European Workshop on Software Process Technology*, pages 70–78.

[PMI, 2000] PMI (2000). A guide to the project management body of knowledge. Technical report, Project Management Institute.

[Ralyté, 2002] Ralyté, J. (2002). Requirements definition for the situational method engineering. In Rolland, C., Brinkkemper, S., and Saeki, M., editors, *Engineering Information Systems in the Internet Context*. Kluwer Academic Publishers.

[Ralyté and Rolland, 2001] Ralyté, J. and Rolland, C. (2001). An assembly process model for method engineering. In Dittrich, K., Geppert, A., and Norrie, M., editors, *Advanced Information Systems Engineering*, number 2068 in LNCS, pages 267–283. Springer-Verlag.

[Rolland et al., 1999] Rolland, C., Prakash, N., and Benjamen, A. (1999). A multi-model view of process modelling. *Requirements Engineeting Journal*, 4:169–187.

[Rupprecht et al., 2000] Rupprecht, C., Funffinger, M., Knublauch, H., and Rose, T. (2000). Capture and dissemination of experience about the construction of engineering processes. In *Proc. of CAISE'89*, number 1789 in LNCS, pages 294–308. Springer.

[Saeki, 2003] Saeki, M. (2003). Came: the first step to automated software engineering. In *Proc. of the OOPSLA'03 Workshop on Process Engineering for Object-Oriented and Component-Based Development*, pages 7–18.

[Scarola and Tatum, 1982] Scarola, J. and Tatum, C. (1982). Definition of project management. In *Proc. of the Specialty Conference on Engineering and Construction Projects*.

[Scott et al., 2001] Scott, L., Carvalho, L., Jeffery, R., and D'Ambra, J. (2001). An evaluation of the spearmint approach to software process modelling. In Ambriola, V., editor, *Proc. of the 8th European Workshop on Software Process Technology*, number 2077 in LNCS, pages 77–89. Springer-Verlag.

[SEI, 1993] SEI (1993). Capability maturity model for software, version 1.1. Technical report, Software Engineering Institute.

[SEI, 2000] SEI (2000). Cmmi for systems engineering/software engineering, version 1.02. Technical report, Software Engineering Institute.

[Serour et al., 2002] Serour, M., Henderson-Sellers, B., Hughes, J., Winder, D., and Chow, L. (2002). Organizational transition to object technology: theory and practice. In Bellahsène, Z., Patel, D., and Rolland, C., editors, *Object-Oriented Information Systems*, number 2425 in LNCS, pages 229–241. Springer–Verlag.

[Stallinger et al., 2003] Stallinger, F., Henderson-Sellers, B., and Torgersson, J. (2003). The oospice assessment component: customizing software process assessment to cbd. In Barbier, F., editor, *Business Component-Based Software Engineering*, chapter 7, pages 119–134. Kluwer Academic Publishers.

[Thomke and Reinertsen, 1998] Thomke, S. and Reinertsen, D. (1998). Agile product development: managing development flexibility in uncertain environments. *California Management Review*, 8(2).

[Thomsett, 2002] Thomsett, R. (2002). *Radical Project management*. Prentice Hall.

[van Slooten and Hodes, 1996] van Slooten, K. and Hodes, B. (1996). Characterizing is development projects. In Brinkkemper, S., Lyytinen, K., and Welke, R., editors, *Proc. of the IFIP TC8 Working Conference on Method Engineering*, pages 29–44. Chapman & Hall.

[van Zyl and Walker, 1982] van Zyl, J. and Walker, A. (1982). Using spice and process innovation to increase capability. In *Proc. of the International Conference on Software Process Improvement and Capability Determination*.

[Yates and Tatum, 1982] Yates, M. and Tatum, C. (1982). Definition of engineering management. In *Proc. of the Specialty Conference on Engineering and Construction Projects*.

[Yourdon, 1999] Yourdon, E. (1999). Software process for the 21st century. *Cutter IT Journal*, 12(9):12–15.

[Yourdon and Argila, 1996] Yourdon, E. and Argila, C. (1996). *Case Studies in Object Oriented Analysis & Design*. Prentice Hall.