



# **21<sup>st</sup> Century Software Development - An “On Demand” Software Engineering Process Perspective**

**Pat O’Sullivan BSc Msc PhD CEng MIEI SMIEEE  
Joe Fitzpatrick BEng CEng MIEI EurIng**

*IBM Dublin Software Lab*

## **Abstract**

This paper examines the implications of an “On Demand” computing model for established software engineering process paradigms. First, an attempt is made to understand the essence of the computing paradigm and the rationale for evolving an “On Demand” computing model. Second, software engineering process methodologies are assessed, after which a summary of peer reviewed research at IBM’s Dublin Software Lab is presented. Finally, we explore recommendations for evolving an “On Demand” engineering agenda in the context of a software engineering process framework.

## **1. Background**

In the early 1980s software was unknown to most of the general public. Large software companies did not exist, computer superstores with vast areas dedicated to the storage of software products were unknown and the Internet was only known to a minority of researchers and academics. In the last few decades this environment has changed. Computer software has become a driving factor in many aspects of the business and economic world.

Books published during the 1970s and 1980s provide historical insight into the changing perception of computers and software and their impact on our business models. Osborne [1979] describes what he considers a new "industrial revolution" while Toffler [1980] calls the advent of microelectronics part of the “the third wave of change” in human history. Naisbitt [1982] predicted the transformation from an “industrial society” to an “information society”. Feigenbaum et al. [1983] suggested that information controlled by computers would be the focal point for power in the twenty-first century and that the “electronic community” created by computer networks was the key to knowledge interchange throughout the world. As the 1980s began, Toffler [1980] described a “power shift” in which old power structures (governmental, educational, industrial, economic, military) disintegrate as computers and software lead to a “democratization of knowledge”. He worried that U.S. companies might lose their competitive edge in global software-related businesses and predicted the demise of the American programmer while

Feigenbaum argued that information technologies were to play a pivotal role in the re-engineering of the global electronic landscape.

On reflection, the IT industry has always been two interrelated industries. The first is computing, and this implies more than chip technologies, database technologies, operating systems, application software and other technology elements that are in a constant state of change. This is more concerned with computing as an architecture, or a model, or a system that integrates all of these disparate components. Computing models do not change very frequently, however a significant change is evolving at present. The second “industry” is the application of computing to improve or transform some aspect of business, where “business” implies the IT challenges of every enterprise and institution. This was not apparent for many years, mainly because these services involved helping IT Managers apply and manage the technology they had purchased. However this second industry was there all along, and was significant. For example, the mainframe model of computing would not have been successful if it resulted in customers simply receiving a new machine. On the contrary, customers had to be introduced to the mainframe business concept, and more importantly IT Managers had to be shown how to apply mainframe systems to transform back end functions like accounting, payroll and inventory management.

## **2. Rationale for an “*On Demand*” Computing Model**

IT managers no longer think of their technology needs just in terms of data centers, or storage systems, or PCs, or even the network. Today, the computing model is seen as the entire technical infrastructure on which businesses run. This is a vital infrastructure that must connect with and support relationships and transactions with other businesses - including devices of all kinds and the various user groups using those devices. This touches on an interesting parallel in terms of how computing is currently being applied by IT managers. In general, businesses and institutions have automated and digitized their standalone operations and processes - the back office, the manufacturing floor, procurement, logistics, customer-facing systems, and so on. They have achieved efficiencies by doing so, and now a need has evolved to transform processes that cut across all of those systems. The challenge going forward is to build a business that can respond dynamically to whatever the world throws at it from an end-to-end perspective. The resulting requirements motivate the evolution of an “*on demand*” computing infrastructure.

In effect, the aspirations of an “*on demand*” computing model for an enterprise or institution is to be able to provide products, services, information, health care, education, government services and so on - in an “*on demand*” way for their customers, citizens, patients and students. These “sense-and-respond” or “real-time” applications have many advantages, as enterprises are able to convert fixed costs into variable costs and they can better manage and control their inventory levels. More compellingly, “*on demand*” computing systems promise greater responsiveness to the needs of customers, employees and business partners. That is obviously very appealing, especially in the current competitive business climate. However, if we consider the magnitude of the business transformation that “*on demand*” computing requires we can see that it is almost as if an organization were turned on its side moving from a collection of vertical “silos” to a seamlessly integrated, horizontal flow across value chains. This represents a substantial shift in business design and in management thinking, and motivates substantial systems and process re-engineering challenges.

Evolving the current computing model to an “*On Demand*” computing paradigm therefore implies new rules for IT infrastructures. Computer systems will need to be integrated and will need to support integration of business processes and operations. More importantly, as IT systems become more complex and expensive the research and development in areas which include Grid Computing, Autonomic Systems,

and open standards becomes more pertinent. The on demand requirements include the need to equip and help IT Managers to build their own internal utilities, software to manage and balance workloads, and server and storage systems to provide additional capacity on demand. This is because today’s organizations look for ways to make their business model more resilient in the face of change and uncertainty. They seek the ability to react to rapidly changing market conditions, manage risk, compete with their competitors with new capabilities, and deliver clear return on investment. This is the key reason why the “on demand” era has evolved, and it represents the next phase of electronic business in which organizations move beyond simply integrating their processes to actually being able to sense and respond to fluctuating market conditions and provide products and services to their customers in an “on demand” way. Electronic business “on demand” encompasses where both computing and business models are converging [IBM, 2002].

So, what are the implications for established software process methodologies in applying an “on demand” computing model? More importantly, what engineering approach should a re-engineering task force consider in terms of addressing the challenges of the new “on demand” computing era?. To address these questions it is significant to note that one key approach to software engineering over the last decade has been the *Process Model*. We will discuss the implications for this in terms of evolving an “On Demand” software engineering approach.

### 3. A Software Engineering Process Perspective

#### 3.1 Introduction

There are various definitions of the term Software Engineering. A concise definition can be found in IEEE [1993] :

1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
2. The study of approaches as in (1) .

The seminal applications of software engineering to software process models is summarized by Humphrey [1989], who defines a software process as “the set of tools, methods, practices and transformations that people use to develop and maintain software and the associated products”. In general, the software process model is defined as a framework for the tasks that are required to build high quality software systems. A software process is typically characterized by a process framework that defines a number of computational activities. While software process models may be constructed at any appropriate level of abstraction, the underlying process architecture must provide the elements, standards, and structural framework for refinement to any desired level of detail.

Fig. 1

Level 1	Level 2	Level 3	Level 4	Level 5
Initial	Repeatable	Defined	Managed	Optimizing

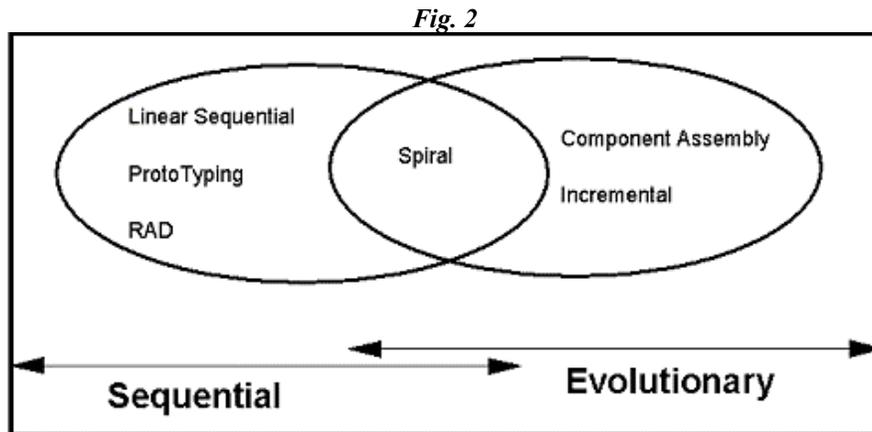
In recent years there has been a significant emphasis on process maturity Paulk [1993]. The Software Engineering Institute (SEI) has established a comprehensive model that is based on a set of software engineering capabilities that need to be present as organizations reach different levels of proficiency. To

determine an organization's current state the SEI uses an assessment questionnaire and a five point grading system (Fig. 1).

The five levels defined by the SEI are derived as a consequence of evaluating responses to the SEI assessment questionnaire that is based on the CMM. The results of the questionnaire are distilled into a single numerical grade that provides an indication of an organization's process maturity. Consequently, the ability to measure and gauge an organization's level is significant, and particularly so for organizations evolving an "On Demand" computing model. This is because the computing evolution has motivated a requirement for a more sophisticated integration of information and business logic, and from different systems and sources. Previously, everyday consumer and business-to-business transactions were hindered by incompatible systems, a lack of standards, and security concerns. Today, the "on demand" business awareness is emerging to address these fundamental barriers. Building the right solution may require a significant understanding of the various systems and business processes, as well as a profound understanding of the various integration points and integration challenges. Understanding process maturity is therefore an important conduit to further improvement and further optimization, both from the perspective of those building the solutions and from the customers who wish to adopt the new computing model.

### 3.2 A Software Engineering Process Paradigm

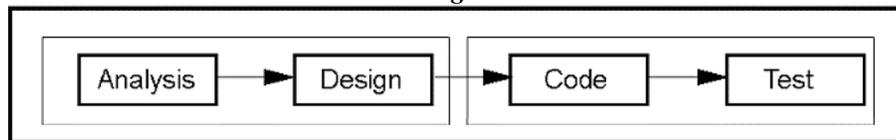
To evolve an on "On Demand" computing model and to apply the model in an industry setting, a software engineer must incorporate an engineering strategy that encompasses the underlying process, methods and tools. This strategy is often referred to as a Software Engineering Paradigm. A software engineering process model is chosen based on the nature of the challenge and the application, the methods and tools to be used, and the control and deliverables that are required. There are a number of Software Process Models in existence today (Fig. 2). The ones most pertinent to applying an "on demand" computing model shall now be discussed.



The most popular process model approach is the *Linear Sequential Process Model* [Royce, 1970]. This model (also called the Classic Life Cycle or Waterfall Model) implies a structured and iterative approach to software engineering that progresses through analysis, design, development, quality assurance and maintenance (Fig. 3). The linear sequential model is by far the oldest and the most widely used process

paradigm in software engineering. However, from the perspective of applying an “on demand” computing agenda there are fundamental criticisms of the model that need to be considered. Firstly, complex real-world projects rarely follow a true sequential flow and a consequence is that the linear model only addresses iteration indirectly. Secondly, the linear sequential model stipulates that all requirements should be stated explicitly and this is often difficult to achieve upfront. A consequence of is that software engineers can become confused when changes are introduced as projects evolve. Ultimately, the linear sequential model has difficulty in accommodating the natural uncertainty that exists at the beginning of many projects. As projects become more complex, then the levels of uncertainty can grow. Thirdly, a working version of the system is generally not available until late in the project cycle, and thus any mistakes can be expensive to rectify. Finally, Bradac et al. [1994] found in an analysis of actual projects that the linear nature of the classic life cycle often leads to significant “blocking states”, in which some project members must wait for others to complete co-dependent tasks. The blocking states tend to be more prevalent at the beginning and end of a project, and the number of blocking states is almost always proportional to the project’s size and level of complexity.

Fig. 3

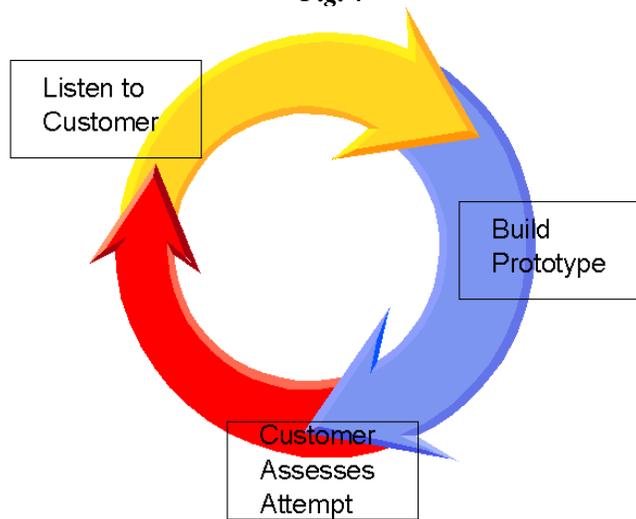


The linear sequential process model can have a definite and important place in building complex solutions and notably when considering an “on demand” application or perspective. It can provide a framework into which methods for analysis, design, development, quality assurance and maintenance can be placed. While the model does have weaknesses it is significantly better than an ad-hoc software engineering approach. However, in planning the use of the linear sequential model some challenges can arise. If implementation teams are able to work with design teams from an early stage then this will overcome many problems, as downstream requirements and considerations can be explicitly stated and planned for. Otherwise late requirements will be difficult to accommodate. The complexities found in the application of an “on demand” computing agenda will necessitate that all requirements and considerations be understood up front.

The *Prototyping Process Model* (Fig. 4) was developed to improve on some of the key shortfalls in the linear sequential model [Bradac et al., 1994]. It begins with requirements gathering - where solutions providers and customers meet and define the overall objectives for the system, identify whatever requirements are known, and outline areas where further definition is required. A quick design then occurs, which focuses on a representation of those aspects of the system that will be visible to the customer. The design leads to the construction of a prototype which is then evaluated by the customer and used as a basis to refine requirements. Iteration occurs as the prototype is tuned, at the same time enabling all parties to gain a better understanding of what is required. However, like the linear sequential model, the prototyping paradigm has a noted weakness with “on demand” applications: The customer sees what appears to be a working version of the system early in the development cycle but may not understand that it is only partially developed. When informed of the need to have the product rebuilt so that high levels of quality can be attained the customer can take the opportunity to request new features. Too often, motivated engineers acquiesce thereby making it more difficult to complete the project within the originally agreed time frame. Also, sub optimal logic, algorithms or even deployment methods may have been chosen to get a prototype ready quickly. The less-than-ideal choice may be kept as a consequence of having to maintain the original schedule with a larger set of requirements. However, although problems can occur, the

prototyping process model can often be an effective paradigm for software engineering as long as the rules of engagement are fully understood on all sides in advance.

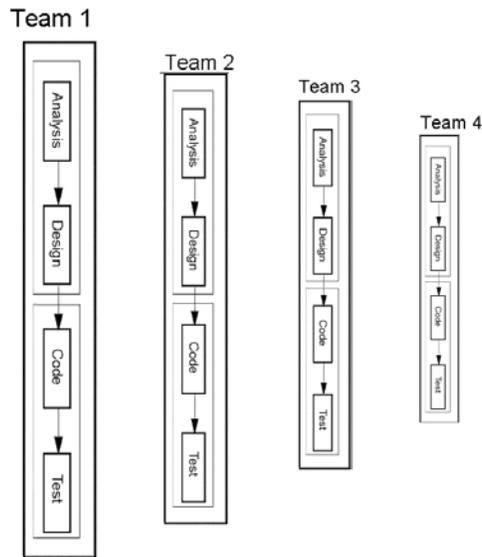
**Fig. 4**



The *Rapid Application Development (RAD) process model* [Butler, 1994] evolved as a solution to the problems caused when issues are discovered late in the development cycle. It is essentially a variant of the linear sequential development process model which emphasizes an extremely short development cycle (Fig. 5). The model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using a component-based construction approach with parallel development and implementation teams. If requirements are well understood and the project scope is constrained, the RAD model's process enables a fully functional system to be created within very short time periods. Used primarily for information systems applications, the RAD approach encompasses phases that include Business Modeling, Data Modeling, Process Modeling, Application Generation, Testing and Turnover. However, the time constraints imposed on RAD projects demand scalable scope, therefore if an engineering challenge can be modularized in a way that enables each major function to be completed quickly and in an isolated way then it may be a candidate for RAD. These modules can be developed and tested separately and then integrated later.

Like the previously discussed process models the RAD approach also has drawbacks [Reilly, 1995], and these are significant in "on demand" applications. Firstly, for large but scalable projects, it requires sufficient human resources to create the right number of teams. Secondly, the model requires developers and customers who are committed to the rapid-fire development activities needed to complete project activities quickly. If commitment is lacking from either side projects will fail. Thirdly, not all types of applications are appropriate for RAD. If early modularization is not an achievable goal then the application of a RAD methodology will force blocking states and dependency considerations. Finally, from an "on demand" perspective, the consequence of adopting the RAD approach can sometimes be negative. This is because the necessity to deliver a fully functional system can quickly cause more subordinate requirements to be deferred until a later stage of the project. This in turn can force substantial changes to the architecture or design, which are both time consuming and expensive.

Fig. 5



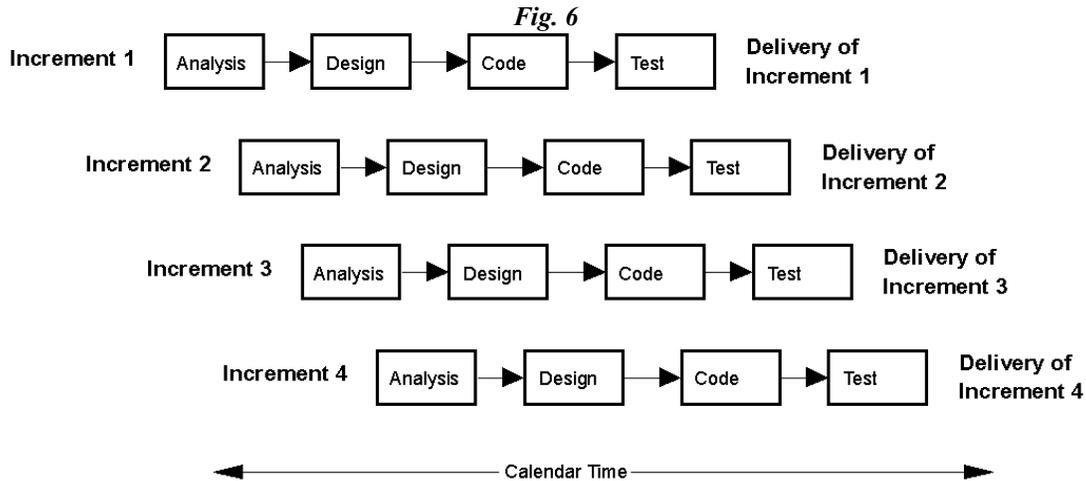
### 3.3 Overcoming Conventional Process Model Limitations

A key limitation with the software process models already discussed is that the typical evolutionary nature of software engineering is not fully considered in conventional software engineering paradigms. This is because these models are iterative and are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software [Boehm, 1988; Davis et al., 1994]. To contend with evolutionary requirements in software engineering, research in evolutionary model theory subsequently evolved. This was essentially owed to a growing recognition that 1) software, like all complex systems, evolves over a period of time [Gilb, 1988], and 2) business and product requirements often change as systems design, development and implementation proceeds for a variety of technical, subjective and non-subjective reasons. Subsequently, software engineers sometimes need a process model that has been explicitly designed to accommodate this evolution. To cope with such challenges a number of evolutionary process models have evolved. The ones that are most pertinent to “on demand” computing applications include the Incremental Model [Basili et al., 1975], the Spiral Model [Boehm, 1988] and the Component Assembly Model [Davis et al., 1994].

The *Incremental Process Model* combines elements of the linear sequential model with the iterative nature of prototyping (Fig. 6). Each linear sequence encapsulates the “analysis + design + code + test” stages to deliver an increment of the software. For example, an email application developed using the incremental paradigm might provide basic email as well as group management facilities in the first increment. More sophisticated email capabilities may be developed in the second increment, email/calendar integration may be added in the third increment, support for non proprietary email protocols in the fourth increment, and so on.

In general the first increment aims to make a solution available that meets the basic requirements but does not contain supplementary features. The customer then reviews this and feedback is then considered in a plan for the subsequent increment. The iterative delivery of each increment is repeated until the completed product is produced.

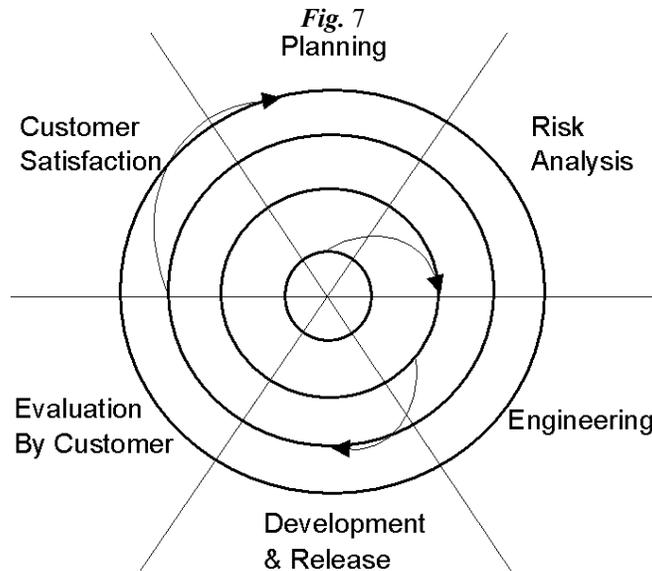
From an “on demand” perspective there can be positive or negative ramifications associated with the adoption of such a model. For example, if key customer requirements are considered as components that must be delayed until a later increment then this is clearly negative, as the risk for fundamental architectural changes is encouraged. On the other hand, the model has tremendous benefit if such concerns are fully considered in the initial increments. Assuming that this is done then subsequent increments can address and refine any areas of concern.



To overcome some of the latter limitations the *Spiral Process Model* and the *Component Assembly Model* evolved as evolutionary software process model to couple the iterative nature of prototyping with the systematic aspects of the linear sequential model and to provide the potential for rapid development of incremental versions of the software (Fig. 7). The spiral model was originally proposed by Boehm [1988] and has an interesting significance for complex “on demand” computing applications, in that the solution or system is developed in a series of incremental releases. During early iterations the incremental release might be a prototype or paper model. Later iterations usually engineer more complete versions of the system.

The component assembly model incorporates many of the characteristics of the spiral model. It is evolutionary in nature and implies an iterative approach to the creation of software. Significantly, the component assembly model involves building applications from prepackaged software components. Object technologies usually provide the technical framework for the building of such components. Components created in other projects can be stored in a library or repository, and then reused. From this it is clear that the component assembly model leads to large software/component reuse, and reusability provides a number of measurable “on demand” benefits — particularly reusability, maintainability, reliability and understandability. The model also allows for reductions in systems development times, quality engineering times, resource counts, project costs, and so on.

From an “on demand” perspective it is significant to note that the use of a component assembly model can be either good or bad. If each of the individual components is designed to meet the customer’s requirements then the benefits are high. However, if some or all of the components fall short in any way then the process of adapting the incorrect component may be complex.



#### **4. Summary**

In general, the type of process model that should be used by software engineers is dependent on a number of factors. First, the complexity of the project may require a more careful and planned approach. In this case a linear sequential or prototyping process approach may be more suitable - as opposed to a more rapid application development approach. Second, the time frame to complete development and testing may be variable. If time is short then a linear sequential approach may not be adequate and a RAD approach may be required. Third, if components are in a mature stage then an evolutionary approach may be suitable. Fourth, the size of the team is also an important factor - if large, then a rapid development approach may be appropriate. If small, then this will not be feasible. Fifth, because the nature of the underlying data will invariably affect the final procedural design, data analysis is as important as process analysis, as data structures are needed to dictate the organization, methods of access, degree of associativity, and processing alternatives for information. Entire texts (e.g. Aho et al., 1983, Kruse, 1984) have been dedicated to these topics.

Adopting a hybrid process model approach may also offer significant advantages where the project elements and its dependencies are well understood. For example, in a typical software development project the user interface design process could follow a RAD approach, and this could facilitate requirements gathering efforts in an incremental and iterative way. The design and implementation of the underlying data model could follow a linear sequential approach, as this would not have the same requirements for iterative customer feedback as the UI model. Also, specific project components (e.g. ancillary services, security concerns) could be readily acquired from expert third party component providers, as these may be seen as non-core competencies. However, adopting a hybrid perspective requires a fundamental competence in terms of understanding the architectural relationships between each of the project's functional and non functional components, as well as a profound understanding of the various integration points and integration challenges. In this regard, understanding process state and process maturity is an important conduit to further improvement and further optimization.

.There are many reasons why the process model may need to be further modified when evolving a new computing agenda. First, a typical project can involve a variety of activities that are carried out by a large number of non-technical internal and external people over large periods of time. When several people work on a common project they need some way to coordinate their work. For relatively small or simple tasks this can often be done informally. With larger groups or more sophisticated activities formal arrangements are needed. As a result significant differences in levels of technical expertise can occur. The process model may need to be modified to cater for these differences and additional steps may need to be added to guarantee quality at all stages. This may result in a degree of repetition of work in coming to terms with isolated or modular components of the project. The process model may need to be modified to cater for this degree of repetition. For example, a typical scenario may include ascertaining the maximum number of parallel activities that can be carried out or determining whether there are any redundant or missing steps. Second, the process model may need to be modified to represent additional project activities and characteristics that are not typically predicted at an outset. Third, an important observation of process management is that process changes adopted in times of crisis are often misguided. These can lead to truncated testing, skipped inspections and deferred documentation. When time is at a premium rationalization is most likely and process judgments are least reliable. Because it is difficult to justify many tests, analyses, or inspections in the heat of the moment, a thoughtfully defined and approved process can be of great help [O'Sullivan et al., 2003].

Before deciding which software engineering process model (or perhaps hybrid) to use in applying an “*on demand*” computing agenda it is important to understand that one of the fundamental problems with software process models in general is that they do not accurately represent the behavioral aspects of what is really done. The main reason for this is that traditional process models are extremely sensitive to task sequence. Subsequently, even simple refinements can result in a complete restructuring of the model being used. Evolving an “*on demand*” agenda will imply working with an inadequate or perhaps legacy architecture, implying a significant refocus of tools, methodologies and process. To address this challenge considerable attention has been devoted to software process modeling [SPW, 1988], however most efforts have focused on the task oriented aspects of processes. Although some behaviorally oriented modeling approaches have been reported [Williams 1988], these efforts still approach modeling from a task orientated perspective. As a result, challenges in evolving a new computing model should not be underestimated and the desired process model may need to be substantially refined to address some of the re-engineering and design complexities. To assist in this challenge it is interesting to note that process technology tools have recently been developed to help engineers analyze the current process, organize work tasks, control and monitor progress, and manage quality [Richardson et al., 1999]. These tools allow an organization to build an automated model of the common process framework, task sets and activities. The model, normally represented as a network, can then be analyzed to determine typical workflow. Alternative process structures that might lead to reduced engineering time or cost can then be examined. Once an acceptable process has been created, other process technology tools can be used to allocate, monitor and control all engineering tasks defined as part of the model.

In Summary, the key drivers for the use of process models in evolving an “*On Demand*” computing agenda are :

- 1) Enablement of effective discussion and communication to analyze (and perhaps critique) the underlying process model being used, prior to considering a new or revised process paradigm.
- 2) Analysis of the existing process model in terms of identifying areas of the process that can be reused in a new or revised process framework.
- 3) Promoting and encouraging management and engineering support for motivating process change and process evolution with a view to evolving a new or revised process model.
- 4) Evaluation of the key aspects of the process model to facilitate ease of process management and change control.
- 5) Ongoing measurement of the various costs and benefits to assess the implications of process changes.

### **Author Biographies**

Pat O'Sullivan is a Principal Engineer at IBM's Dublin Software Lab. He completed his MSc in 1997 and his PhD in 2001, and both of his IBM sponsored research projects won industry awards. Pat has over 12 years Software Engineering experience with the latter 9 years of these at IBM. His research and publications interests include HCI, Software Engineering/Re-Engineering, UIMS, Software Globalization and Test Automation. Pat is a Senior Member of the IEEE and he is a Chartered Engineer with the Institute of Engineers of Ireland. He is also a founding and committee member of the newly formed SoftTest Ireland SIG and is an IBM industry expert representative for the European sponsored ELECT project.

Joe Fitzpatrick is a Software Executive at IBM's Dublin Software Lab in Ireland. He graduated from University College Dublin with a B.Eng. in Mechanical Engineering in 1982. Joe has 19 years Software Engineering experience initially in the automotive industry followed by 15 years with Lotus Software/IBM Software Group. As part of the original Lotus Software Globalization Team, Joe defined and documented many of Lotus' internal best practice guidelines for designing internationalizable software. Joe is a Chartered member of the Institute of Engineers of Ireland (IEI) and also holds the FEANI title of EurIng. He is a committee member of the Information and Communications Technology Division of the IEI.

# References

1. Aho, A.V., Hopcroft, J., Ullmann, J., *"Data Structures and Algorithms"*, Boston, MA: Addison-Wesley, 1983
2. Basili, V.R., Turner, A.J., *"Iterative Enhancement: A Practical Technique for Software Development"*, IEEE Transactions on Software Engineering, SE-1-4, Dec. 1975
3. Berzins V., Gray M., Naumann D., *"Abstraction-based software development"*, Communications of the ACM, Vol. 29, No. 5, May 1996, pp.402-415
4. Boehm, B., *"A Spiral Model for Software Development and Enhancement"*, Computer, Vol. 21, No. 5, May 1988, pp.61-72
5. Bradac, M., Perry, D., Votta, L., *"Prototyping a Process Monitoring Experiment"*, IEEE Transactions on Software Engineering, Vol. 20, Issue No. 10, Oct. 1994, pp.774-784
6. Butler, J., *"Rapid Application Development in Action"*, Managing System Development, Applied Computer Research, Vol. 14, No. 5, May 1994, pp.6-8
7. Davis, A., Sitaram, P., *"A Concurrent Process Model for Software Development"*, Software Engineering Notes, ACM Press, Vol. 19, No. 2, pp.38-51, Apr. 1994
8. Feigenbaum, E.A., McCorduck, P., *"The Fifth Generation"*, Boston, MA: Addison-Wesley, 1983
9. Gane, C., Sarson, T., *"Structured Systems Analysis: Tools and Techniques"*, Englewood Cliffs, Upper Saddle River, NJ: Prentice Hall, 1979
10. Gilb, T., *"Principles of Software Engineering Management"*, Boston, MA: Addison-Wesley, 1988
11. Humphrey, W.S., *"Managing the Software Process"*, Addison-Wesley, Reading, MA: 1989
12. IEEE, *"Standards Collection: Software Engineering"*, IEEE Standard 610.12-1990, IEEE, 1993
13. Kruse, R.L., *"Data Structures and Program Design"*, San Diego, CA: Prentice-Hall, 1984
14. Naisbitt, J., *"Megatrends"*, Warner Books, 1982
15. O'Sullivan, P., Fitzpatrick, J., *"The Implications for an On Demand Computing Model for Established Software Engineering Process Paradigms"*, Paper recently submitted to the IBM Center for Advanced Studies Conference 2003, IBM Canada
16. Osborne, A., *"The next Industrial Revolution"*, New York, NY: McGrath Hill, 1979.
17. IBM Annual Report, 2002
18. Paulk, M., *"Capability Maturity Model for Software"*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993
19. Richardson, I., Ryan, K., Murphy, E., *"Improving the Software Process in Small Indigenous Software Development Companies Using a Model Based on Quality Function Deployment"*, PhD Thesis, University of Limerick, 1999
20. Royce, WW., *"Managing the Development of Large Software Systems: Concepts and Techniques"*, Proceedings of IEEE WESCON, IEEE, August 1970, pp. 1-9
21. Ross, D.T., Schoman K.E., *"Structured analysis for requirement definition"*, IEEE Transactions for Software Engineering, SE-3(1), 1977
22. SPW (Software Process Workshop), Fourth International Software Process Workshop, *"Representing and Enacting the Software Process"*, Moretonhampstead, Devon: UK, May 11-13, 1988
23. Stefik, M., Babrow, D., *"Object Oriented Programming: Themes and Variations"*, A.I. Magazine, Volume 6, No. 4, p.41

*The Implications of an “On Demand” Computing Model for Established Software Engineering Process Paradigms*

24. Toffler, A., *“The Third Wave”*, New York: NY, Morrow, 1980
25. Williams, L.G., *“Software Process Modeling: A Behavioural Approach”*, Proceedings of the 10th International Conference on System Engineering, IEEE, 1988, pp. 174-186