E1. PROJECT TITLE:

Size Measurement and Effort Estimation of Complex Real-Time Systems

E2. Aims and Background

Software involved in many complex systems such as aircraft navigation, radio telescope control and patient monitoring has real-time constraints. Temporal constraints, hardware interaction and concurrency introduce considerable complexity in their development. This makes estimation of effort required to develop such systems a great challenge. ***This project addresses the challenge of reliable effort estimation in developing complex real-time systems.***

Complex systems involve huge investment, and it is essential to have reliable effort estimation methods to be able to develop them in a cost effective manner. Unrealistic project deadlines, often caused by inaccurate effort estimation has caused delays and cost blowout for any number of real-time systems projects. A famous example is the Denver airport automated luggage-handling system causing 16 months delay in opening the new airport and US$ 1 million loss every day of the delay.

E2.1 Real-Time Systems

A real-time system is one in which **time**[*] is a criterion in determining the **correctness** of the behaviour of the system (Marz and Plakosh, 2001). There are generally three factors that distinguish a real-time system from most other software systems. They are:

1) temporal properties and constraints that the system must satisfy
2) concurrency (i.e, the need for multiple processes to communicate and coordinate their operations), and
3) interaction between hardware and software

For example (from Douglass, 1999), a remote controlled robot used in hazardous situations such as defusing a bomb or handling dangerous chemicals may be required to move its arm closely with the arm movement of a person wearing an "electronic" glove standing far away. The software part of the solution needs separate processes, one to track the person's glove movement and another to perform the robot's arm movement. Furthermore, the robot's arm movement must be quick enough to take advantage of the person's inherent motor-control system. If the robot responds too slowly, the person may overreact thus creating an unstable control loop. If we assume a maximum finger movement speed of 10 cm/sec, the robot must track a linear motion with no more than 50 ms delay.

The complexity that the three factors bring into the development process is significant and not well understood. Thus, effort estimation techniques used for sequential software systems are often insufficient to handle real-time systems. Putnam and Myers (1997) in a study of 3885 projects observed that effort required for a real-time system was always greater than the effort required to develop a business system of the same size.

[*] Timeliness is next to Godliness – Douglass' law

E2.2 Effort Estimation

To a great extent, development effort depends on the characteristics of the software developed; it is common to estimate effort (or project cost) from a measure of *Software Size* (Ebrahimi, 1999). For example, *COCOMO* (COnstructive COst MOdel) (Boehm, 1981) estimates Effort E using the function:

$$E(s) = as^{b}f,$$

where s is the software size measured in Kilo Delivered Source Instructions, f is an adjustment factor, and the coefficients a and b are determined from the type of software constructed.

E2.2.1. Function Points and Full Function Points

In order to estimate effort, therefore, we need to have a reliable size measure for real-time systems. The size measure that is widely used in the industry (next to Lines-of-Code count) is *Function Points* (Albrecht, 1979; Jeffery and Lo, 1997). The function point value is determined by classifying the functional elements of the requirements into areas such as external input, external output, etc. For each classification, the International Function Point Users Group Manual provides a weight. The Function Point value is calculated as the sum of all the relevant weights multiplied by a Value Adjustment Factor determined by the application type.

The function point technique is very much biased towards transaction-based systems. In other systems such as compilers, control systems, and CAD systems, measuring size based on inputs and outputs are not sufficient; much of the complexity is hidden in the algorithms, data structures and architectures. Jeffery, Lo and Barnes (1993), Fenton and Pfleeger (1997) and others have pointed out practical and theoretical shortcomings of Function Points. On the other hand, since Function Points can be measured from a requirements document, the value is available early in the Software Life Cycle, unlike size measures such as Lines-of-Code. Therefore Function Points would allow estimation of effort early in the life cycle. Extensions and modifications proposed to Function Point techniques include Feature Points (Jones, 1991), which takes algorithms in its count, and 3D Function Points (Whitmire, 1992), which counts internal operations that transform input to output data. Among these, *Full Function Points* (Abran et al., 1997) is particularly designed for real-time systems.

The Full Function Points (FFP) method counts functional transactions in the requirements specifications; these transactions can be with an end user, a mechanical device or another application program. FFP counts data movements within the application (Read and Write) and across the boundary of the application (Entry and Exit). FFP recognizes that Real-Time Systems often require hardware-software interaction. However, it does not seem to take into account the influence of temporal and concurrency constraints which may not be reflected as data movements in the specification.

Besides, as the authors of FFP admit (Abran et al., 2001), it is not yet capable of handling software that implements complex algorithms or other complex rules and constraints. Furthermore, FFP seems to have the same shortcomings as Function Points in satisfying the scientific theories of measurement. To our knowledge no work has been reported demonstrating the theoretical validity of FFP.

“Software Engineering” is still in many aspects a craft. While significant progress has been made in the understanding of software measurement, the complexity of problems that require software solutions has increased over the years. This "moving of the goal post" has been an additional impediment for measurement experts to develop measures that satisfy the scientific principles of measurement and are empirically validated. Software Engineering being a relatively young discipline, is experiencing changes and innovation in a much more rapid fashion than some of the mature fields of engineering. Change introduces new attributes and properties, and consequently new measures need to be developed to understand and use them.

This project aims to improve effort estimation of real-time software by developing better effort prediction models based on sound and valid software size measures.

We will use the approach of estimating effort from size, however, our aim is to locate all significant variables that influence effort and measure their properties in calculating software size.

E3. Significance and Innovation

Reliable prediction of effort is still one of the great challenges of Software Engineering. The abstract nature of software, unavailability of well-formed measures and recurring changes in requirements during the development process make accurate measurement of properties and reliable prediction of effort very difficult.

The aims of this project are to:

- ***develop a well-formed <u>software size measure</u> and an <u>effort prediction model</u> for complex real-time software systems, and***
- ***formally and empirically <u>validate</u> the measures.***

E3.1 Software Effort Measures

From the mathematical point of view, a cost measure for a software project consists of a function

$C: D \rightarrow R,$

where D is a domain of software projects and R is the positive real numbers, so that for a given project d, $C(d)$ is the cost of d. The function should satisfy some fundamental mathematical rules; for example, if d_1 and d_2 are two disjoint "subprojects" and d is the "combination" of d_1 and d_2 then $C(d) < C(d_1)+C(d_2) + H$, where H is a constant. The cost is often indirectly determined from a direct measure of Software Size S:

$S: D \rightarrow R.$

An effort measure

$E: R \rightarrow R,$

such as the COCOMO model above, can be used to map the software size $S(d)$ to effort.

This project is concerned with both the map from the <u>project to software size</u>, and the map from <u>software size to effort</u> in the domain of real-time systems.

E3.2 Using the Unified Modeling Language (UML)

Selic (1999) explains that the object-oriented paradigm is an excellent fit to Real-Time Systems since in such systems the structure (i.e., the system) is constant and the behaviour (i.e., functions) necessarily needs to change with time. For a model to be predictive, however, it is not sufficient to model the structure and behaviour of the software, but also we need to model the logical and physical resources the software is dependent on (Selic, 1999). This requirement distinguishes modeling real-time systems from other software systems. Recently, the *Unified Modeling Language (UML)* has emerged as the industry standard for requirements modeling and design of object-oriented systems. UML is now accepted as the design standard by the Object Management

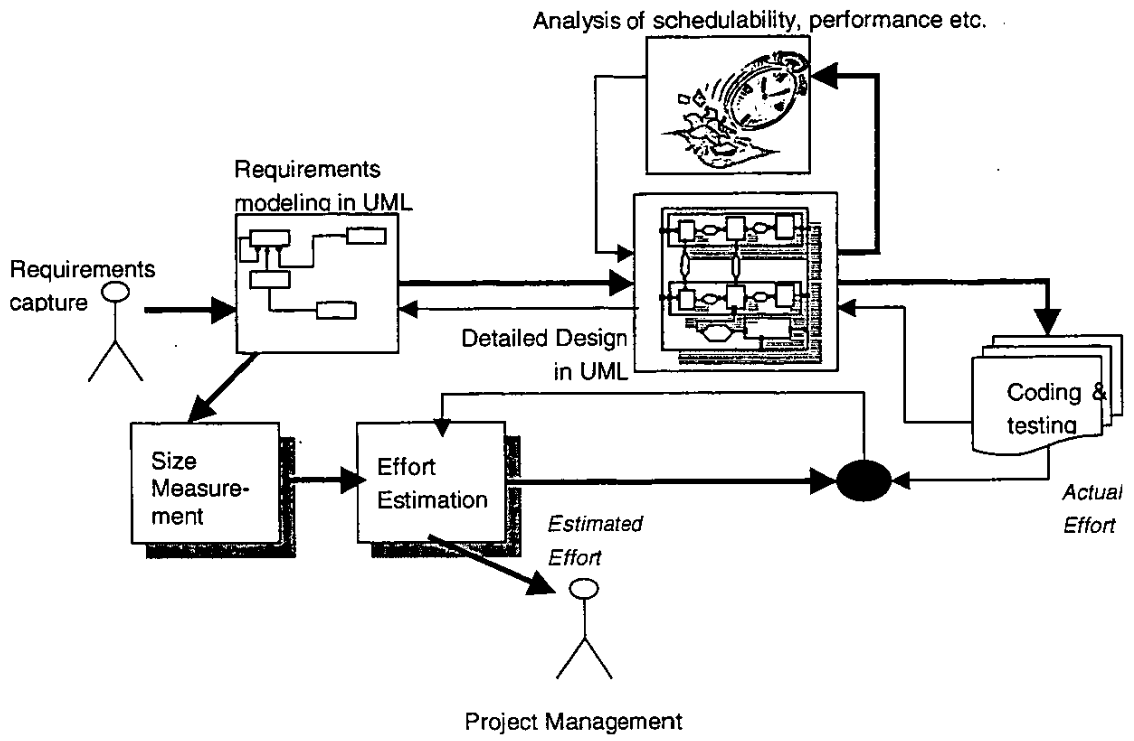


Figure 1. Measurement and estimation in real-time software life cycle
(Thick arrows show main flow of information; thin arrows show feedback)

Group, a consortium of over 2500 world-wide IT institutions, including companies such as Microsoft, IBM, Sun, Cisco, Hewlett-Packard, Oracle, Motorola, Lucent, Fujitsu, SAP, and Object-Oriented. UML is widespread and provides formalism through aspects such as Object Constraint Language (OCL); OCL facilitates precise specification of the semantics of the requirements. It opens the possibility of developing scientifically rigorous size measures that are useful to the worldwide community of software engineers.

We propose to investigate the use of UML artifacts as the domain for size measures.

The benefit of UML is that not only it is useful in capturing and modeling the requirements in a precise manner, but the same notation can be used to complete the design from the captured requirements as shown in Figure 1. The Figure also shows how Size Measurement and Effort Estimation fit into the software life cycle; size measurement is done directly from the requirements model. Our hypothesis is that the effort required to develop a requirements specification using UML is not significantly greater than what is required for a natural language specification, but using UML and its associated formal notations such as Object Constraint Language will significantly improve the accuracy of estimating post requirements effort.

Several approaches to modeling real-time systems using UML have been proposed. Examples are McLaughlin and Moore (1998), Selic and Rumbaugh (1998), and Douglass (1998). Many of these approaches are combined into the Object Management Group's (2002) profile for modeling key aspects of real-time systems. The profile emphasizes schedulability, time and performance specification. At the core of the framework is the notion of Quality of Service (QoS), which provides a uniform basis for attaching quantitative information to UML models. The profile uses

UML stereotypes[1] to derive new (meta) classes for handling real-time requirements. Examples are RTtime, CRconcurrent (a concurrent unit), SAction (a scheduling behaviour characterized by its own required Quality of Service characteristics) etc. The report points out that the techniques have been already used successfully in several real-time systems projects. *This gives us further confidence in using UML to measure software size.*

E3.4 Multidimensional Size Functions

Large real-time systems need hundreds, even, thousands of components. Software Engineers need to know the size of components, subsystems as well as the whole system. Further, there are different aspects to the size function such as functionality and complexity of the system. Henderson-Sellers (1996) discusses computational, representational, problem and structural complexities in addition to programmer characteristics. The implication is that the size function has a *multidimensional range*:

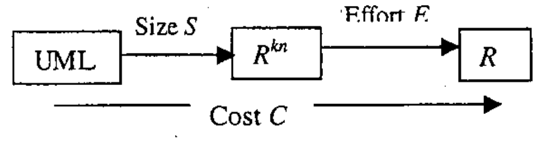
$$S: D \rightarrow R^{kn}.$$

Here the domain D is composed of subprojects of some kind (for example, UML documents describing classes or components), and R^{kn} is a set of k-tuples of n-tuples of real numbers. The jth element of $S(d)$ is the size of the jth subproject. This size is measured as an n-tuple of real numbers, each measuring an aspect. To estimate the total cost of a project, the engineer needs an effort model

$$E: R^{kn} \rightarrow R,$$

and the cost function is the combination of S and E.

The summary of our approach is in Figure 2. Here the domain is UML, the size is a vector of length kn, and the cost is the composition of the size and effort functions.



$$\begin{tikzcd} \boxed{\text{UML}} \arrow[r, "\text{Size } S"] \arrow[rr, bend right, "\text{Cost } C"'] & \boxed{R^{kn}} \arrow[r, "\text{Effort } E"] & \boxed{R} \end{tikzcd}$$

Figure 2

E3.5. ARC's Designated Priority Area of Research

The Australian Research Council has designated Complex Systems as a priority area. Among the indicated fields of research (Australian Research Council, 2002), this project comes under *Software Engineering* of Complex Systems. Just as it is "difficult to understand, predict and manage" the behaviour of complex systems, it is also difficult to predict and manage the development processes of systems with such complex behaviour. This research project will enhance the degree of understanding of the effort required to produce complex software systems that have real-time constraints, hardware-software interaction and concurrency; this will be achieved by developing and validating measurement and effort estimation models for such systems.

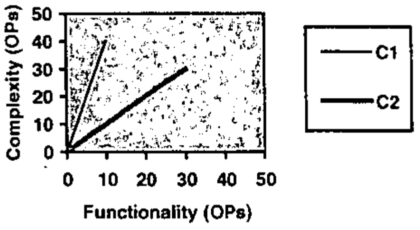
E4. Approach

Our methodology is primarily empirical: we will design and then test size measures and effort estimation models. We will design a range of hypotheses for measures, apply them to data (see **Empirical Evaluation** below), and compare the measures to the actual project efforts.

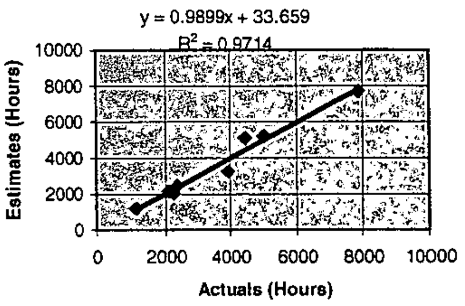
[1] A stereotype is derived from an existing meta-class in the UML metamodel. It's an elegant way of extending the modeling language for specific needs.

We intend to develop objective measures for Software Size *S*, based on the internal characteristics of real-time systems. In order to compute effort *E* from software size, we will develop a prognostic model (Henderson-Sellers, 1995) that is empirically testable.

To develop these models, we will analyze the key characteristics of real-time systems in order to identify the parameters of a size measure. We will avoid failures of measures such as Function Points by taking various complexity factors into consideration. Our earlier work in this regard (this year funded by an ARC-discovery grant) studied sequential systems, but emphasized their problem complexity (Hastings and Sajeev, 1997; Hastings and Sajeev, 2001). We expressed Size as a two dimensional vector of functionality and problem-complexity (Graph 1) measured in terms of operational units (OPs) in the signature and axiom sections, respectively, of algebraic specifications. In Graph 1, we have two vectors representing the size of two components; C2 provides lesser functionality, but has a higher complexity than C1.



Graph 1. Vector Size Measure



Graph 2. Actual Effort vs. Estimated Effort

We will identify the different aspects of the Unified Modeling Language and determine how they contribute to the size of the software. The Unified Modeling Language provides several features to express the semantics of the requirements. The Object Constraint Language (OCL) associated with UML is used for specifying constraints of the various elements of the system. UML has several features, from use-cases to Statecharts, to specify the behavior. The recent Real-Time profile produced by OMG introduces real-time specific stereotypes. We will extract various complexity factors from the specification of system behavior and constraints. By formulating a measure of these characteristics, we will be able to calculate the complexity component of software size.

For software engineers, a primary measure such as Software Size is useful for project planning, and, significantly, to estimate secondary measures such as Effort. We aim to develop a prediction model for effort estimation based on Software Size. The model will take into consideration the fact that the effort required for a component is not entirely dependent on its size in a component based system; it is also dependent on the nature of the component (e.g., whether it is custom built or being reused). The effort required to search component libraries, integrate components etc. also need to be taken into consideration. We will study the parameters involved in these variations, and derive cost-drivers based on empirical data.

In our initial study[2] we used two cost drivers: the magnitude and gradient of the size vector. An experiment involving eight industrial projects showed a strong correlation between estimated and actual effort (See Graph 2).

We will conduct theoretical and empirical validation of the size measure and effort prediction model:

1. **Theoretical validation** to ensure that the measure is in accordance with measurement theory will be conducted within the framework of Kitchenham *et al.* (1995). The Kitchenham framework identifies four criteria that need to be satisfied:
   - For an attribute to be measurable, it must allow different entities to be distinguished from one another.
   - A valid measure must obey the Representation Condition. The representation condition is that: "To measure the attribute we need to have corresponding relations in some number system; then measurement is the assignment of numbers to the entities in such a way that these relations are preserved" (Fenton and Pfleeger, 1997).
   - Each unit of an attribute contributing to a valid measure is equivalent.
   - Different entities can have the same attribute value (within limits of measurement error).

2. **Empirical evaluation.** We will collect data from two sources. One is sources that are reported in the literature (e.g., Putnam and Myers (1997)). The other is Australian software companies that develop real-time systems. BAE Systems (one of the largest aerospace companies in the world after the merger of British Aerospace and GEC Marconi Systems) has significant operations in Williamtown in Newcastle. BAE Systems at the local level has agreed to make available parts of their projects to test our effort prediction models[3]. We are negotiating with two other companies (one in health monitoring systems and the other in control systems) in order to increase the diversity of industrial projects from which data is collected.

   Multiple regression analysis will be used for empirical evaluation. Data (requirements and design documents, and documentation of actual effort on all project tasks) will come from project databases. We are particularly interested in using regression analysis to determine which aspects of size contribute most to the total overall effort.

   We will conduct two levels of empirical testing: an initial post-hoc empirical evaluation, and a full empirical validation.

   An initial empirical evaluation will involve post-hoc (after the fact) statistical analysis of completed projects. This cannot confirm a causal relationship, but is a quick and appropriate method for a preliminary study. Our main emphasis will be to compare our approach with other measures such as Full Function Points. Software requirements will form the baseline for the validation. A measurement tool will be developed to assist in measuring the attributes from UML requirement specifications. For each project, the requirements (if not already in UML) will be reengineered into UML and product attributes will be measured. The process attributes (effort, duration, team size etc.) will be collected from project documentation.

[2] This model was developed for traditional systems and based on algebraic specifications; see (Hastings and Sajeev, 2001).

[3] This consent, as is usual in such cases, is subject to security clearance and corporate clearance.

Based on the outcomes of the initial evaluation, we will re-evaluate the model and make any modifications necessary. Then, a full empirical validation is planned in cooperation with industries mentioned above. We will track appropriately scoped projects (or defined parts of it) from requirements modeling to project completion. We will do statistical analysis to determine correlation between actual and predicted effort, their statistical significance and error margin.

E4.1 Timing

The tasks involved in the project are:
T1: ***Analysis*** of characteristics of real-time systems and their relations in UML
T2: ***Collection*** of preliminary project ***data***
T3: Design of ***size measure***
T4: Development of ***effort estimation model***
T5: ***Theoretical validation*** and initial empirical testing
T6: ***Thesis*** completion
T7: ***Reevaluation*** of models
T8: Empirical ***validation***

The table below shows the timing and people involved in various tasks. Apart from the chief investigator (CI) there is a PhD student and a research assistant (RA).

Half Year	**Tasks**							
	T1 Analysis	**T2** Data Collection	**T3** Size Measure	**T4** Effort Model	**T5** Initial Validation	**T6** Thesis	**T7** Model Re-evaluation	**T8** Empirical Validation
2003/1	CI, PhD	RA, CI						
2003/2			CI, PhD					
2004/1			CI, PhD	CI, PhD				
2004/2				CI, PhD				
2005/1					CI, PhD			
2005/2					CI, PhD	PhD, CI	CI	CI, RA
2006/1								CI, RA
2006/2								CI, RA

E5. National Benefit

E5.1 Expected Outcomes

- A new size measure and prediction model for Complex Real-Time Software Systems

- Empirical evaluation of size measure and prediction model
- A PhD graduate with expertise in Software Metrics and Real-Time Systems
- Refereed journal and conference publications

E5.2 Benefit to Economy

Software is a key element of the US$ 2 trillion global spending on Information and Communications Technologies (ICT) (World Information and Technology Services and Alliances, 2000). Complex software systems are often the costliest to develop. Unreliable models for estimation of development effort of such systems have cost the economy dearly. For example, when the software behind the tolling system in Melbourne's CityLink project was delayed, it cost the toll way operator $1.4 million a week in lost revenue; motorists had a free ride on the newly built road system for months (Australian Broadcasting Corporation, 1999).

Successful completion of this project is expected to significantly improve the reliability of estimating development effort for complex real-time software systems, thus reducing cost explosion from over-promising and under-resourcing. Since Australia is one of the top ten countries in the world (in both GDP and per-capita terms) in Information and Communications Technologies spending, benefit to Australian economy will be proportionately high.

E6. Communication of Results

The methods and techniques will be made available to Australian Software industry (in particular, partners who supply the data for testing) to adopt in their software process.

The results will be published in national and international journals and conferences thus giving international recognition for an Australian research outcome.

E7. Description of Personnel

The chief investigator will take intellectual responsibility for the project, its conception, all strategic decisions, and the communication of the results. In all stages, he will provide the research directions and ideas.

Tasks T1 (Analysis), T4 (Size Measurement), T7 (Model Revaluation), which have a large conceptual component, will be investigated by the CI.

The CI will supervise a research assistant in data collection and storage aspects (Task T2).

The sequence of tasks T3 - T6 (Size Measure, Effort Estimation Model, Theoretical Validation and Empirical Testing) has the right scope for a PhD project. Task T7 (Empirical Validation) will be conducted by the CI with the help of a research assistant; the RI will collect the data, do periodic monitoring of progress and help the CI do the analysis.

E8. References

Abran, A., Maya, M., Desharnais, J. M. and St-Perre, D. (1997): Adapting Function Points to Real-Time Software, American Programmer, 10(11), pp. 32-43.

Abran, A. et al. (2001): *COSMIC Full Function Points, Version 2.1*, Common software measurement international consortium, www.cosmicon.com/

Albrecht, A. J. (1979): Measuring application development productivity, in *Proceedings of IBM Applications Development Symposium*, 83-92.

Australian Broadcasting Corporation (1999), Melbourne's Citylink Tollway Near Completion, *7.30 report*, October 24.

Australian Research Council (2002): *Discovery Projects: Guidelines for Applicants for Funding Commencing in 2003*, www.arc.gov.au/ncgp/discovery/projects/default.html

Boehm, B. W. (1981): *Software Engineering Economics*, Prentice-Hall.

Douglass, B. P. (1998): *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Addison-Wesley.

Douglass, B. P. (1999): Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns, Addison-Wesley.

Ebrahimi, N. B. (1999): How to Improve Calibration of Cost Models, *IEEE Transactions on Software Engineering*, 25(1), Jan/Feb.

Fenton, N. E. and Pfleeger, S. L. (1997): *Software Metrics: A Rigorous and Practical Approach, 2nd Edition*. London: PWS Publishing Company.

Hastings, T. E. and Sajeev, A. S. M. (1997): A Vector Based Software Size Measure, in *Proceedings of the Australian Software Engineering Conference*, Sydney, Australia, IEEE Computer Society, 7-15.

Hastings, T. E. and Sajeev, A. S. M. (2001): A Vector Based Approach to Software Size Measurement and Effort Estimation, *IEEE Transactions on Software Engineering*, 27(4), April, pp. 337-350.

Henderson-Sellers, B. (1995): OO Metrics Programme, *Object Magazine*, Oct., 73-79.

Henderson-Sellers, B. (1996): *Object-Oriented Metrics: Measures of Complexity*, Prentice Hall.

Jeffery, D. R., Low, G. C. and Barnes, M. (1993): A Comparison of Function Point Counting Techniques, *IEEE Transactions on Software Engineering*, 19(5), 529-532.

Jeffery, D. R. and Low, G. C. (1997): Function Points and their Use, *Australian Computer Journal*, 29 (4), 148-156.

Jones, C. (1991): *Applied Software Measurement*. New York, NY: McGraw-Hill.

Kitchenham, B., Pfleeger, S. L. and Fenton, N. (1995): Towards a Framework for Software Measurement Validation, *IEEE Transactions on Software Engineering*, 21(12), 929-944.

McLaughlin, M. J and Moore, A. (1998): Real-Time Extensions to UML, Dr. Dobb's Journal, December.

Marz, T. F. and Plakosh, D. (2001): Real-Time Systems Engineering: Lessons Learned from Independent Technical Assessments, Technical Note, CMU/SEI-2001-TN-004, Software Engineering Institute.

Object Management Group (2002): UML Profile for Schedulability, Performance and Time-Specification, ptc/2002-01-20, www.omg.org

Putnam, L. H. and Myers, W. (1997): Industrial Strength Software: Effective Management using Measurement, IEEE Computer Society Press.

Selic, B. (1999): Turning Clockwise: Using UML in the Real-Time Domain, *Communications of the ACM*, 42(10), October, pp. 46-54.

Selic, B. and Rumbaugh, J. (1998): Using UML for Modeling Complex Real-Time Systems, *Whitepaper*, Rational (www.rational.com/products/whitepapers/UML-rt.pdf)

Whitmire, S. A. (1992): 3D Function Points: Scientific and Real-Time Extensions to Function Points, Proc. Pacific Northwest Software Quality Conference.

World Information and Technology Services and Alliance (2000), *Digitial Planet 2000: The Global Information Economy Executive Summary*, www.witsa.org, 2001.