Rapid Developmnt of High-Quality Customizable and Adaptable Software for Digital Signal Processors

Farokh B. Bastani[1], John Linn[2], Kashi Rao[3], I-Ling Yen[1], Simeon Ntafos[1]

Abstract

Dramatic advances in computer and communication technologies have greatly promoted the growth of embedded telecommunication systems. The software for these applications is becoming increasingly sophisticated and complex and this trend will accelerate over the next few years with the development of "software-defined telephony". We present an approach for developing rigorous techniques for rapidly constructing highly dependable embedded applications. It emphasizes the use of standard software frameworks, down-loadable and updatable code modules, and commercial-off-the-shelf components to construct complex and dynamically changing embedded software systems. Our domain-specific focus is used to leverage the special characteristics of real-time embedded programs to develop deep knowledge bases, tools, and techniques for achieving accelerated development schedules and high-quality assurance.

1. Introduction

Dramatic advances in computer and communication technologies have greatly reduced hardware costs and improved their performance and reliability. This has made it economically feasible to extend the reach of automation to more and more critical services, such as banking and financial services, remote patient monitoring systems, transportation, etc. The market for these embedded computer systems is huge and is expected to grow rapidly over the next few years. For example, 260 million cellular phones were sold in 1999, and this pace will continue with the introduction of new 3G cellular infrastructure starting in 2001. Some other emerging embedded computer applications include television/settop box and blending of TV & Internet, Internet and digital communications infrastructure, and residential gateway and Internet in the home.

Meanwhile, software continues to become more and more complex due to the growing sophistication and complexity of modern applications. For example, consider telecommunication systems. Just a few years ago, all that a switching system had to do was to establish a route for a call, monitor the call for billing purposes, and release the resources dedicated to the call after it was completed. In recent years, this simple scenario has become extremely complex with an explosive growth in the number of features and capabilities. Telecommunications systems must now handle stationary and mobile calls (both cellular and satellite wireless systems), handle various failure modes (switches, trunk-lines, satellites), support voice and data transmissions, handle different service plans, and provide numerous user-oriented features (call forwarding, speed dialing, caller id, 911 service, etc.). The role of software in telecommunication systems is expected to explode dramatically over the next few years with the development of “software-defined telephony.” These smart mobile phones will shift communication functions to programmable components in order to add features and functions on-the-fly and adapt instantly to different frequencies and transmission standards.

At the same time, telecommunications systems are becoming crucial links in mission-critical applications (banking, stock-exchange, electronic commerce, etc.) and even safety-critical applications (tele-medicine services, defense systems, early warning systems, etc.). For these critical applications, it is necessary to be able not only to achieve high quality but also to rigorously *demonstrate* that high quality has in fact been achieved. In today’s highly competitive business environment, it is also essential to have accelerated development schedules to exploit windows of opportunity. Furthermore, success in today’s global marketplace requires the capability to quickly customize and adapt products for niche markets and to satisfy diverse regional standards and procedures.

To meet all these challenges, software development technology is rapidly shifting away from low-level programming issues to automated code generation and integration of systems from components, either Commercial-Off-The-Shelf (COTS) components or specially developed in-house components. This is made possible by numerous recent breakthroughs in software technology, including web-based cooperative software development, in-process monitoring, agents, Java, scripting languages, and, especially, industry-driven standardization efforts, such as CORBA, TINA, TL 9000, and XDAIS. The use of COTS components can significantly reduce software development time and cost. However, the downside is loss of control over the quality of the system, especially with the use of third party software components. Rigorous techniques for rapidly constructing highly reliable software for embedded systems must emphasize modular design & software engineering principles, use of standard software frameworks, downloadable and updatable code modules, and complex and dynamically changing software configurations.

The Embedded Software Center is developing APEX (Advanced Programming Environment for Embedded Computer Systems), an infrastructure for enabling the rapid development of embedded software from third party and COTS components. APEX is a distributed web-based.

[1] Department of Computer Science MS EC-31, University of Texas at Dallas, Richardson, TX 75083-0688.
[2] Texas Instruments, Systems and Software Lab.
[3] Alcatel Corporate Research Center, Richardson

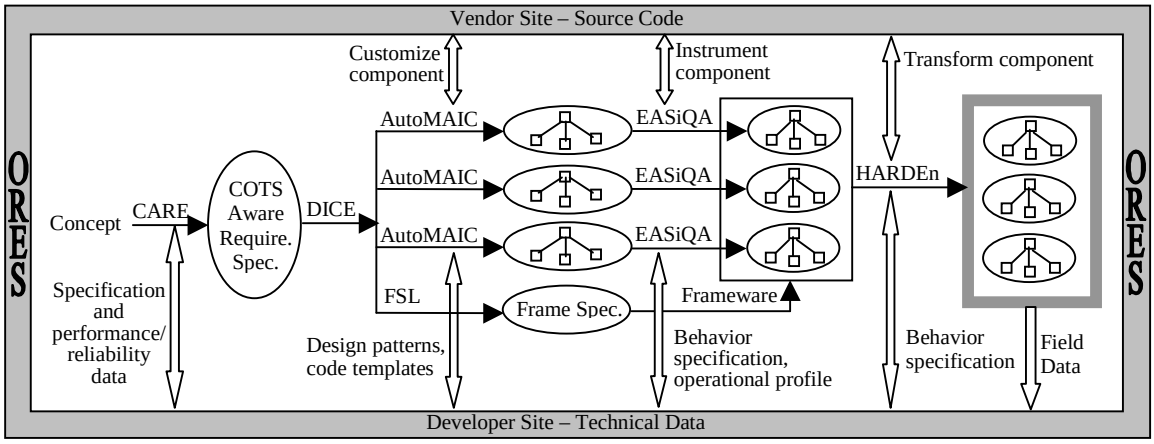

Fig. 1. The APEX Infrastructure.

system that helps acquire data for software components, customize the components, integrate them together, perform testing and quality assurance, and enhance the security and fault tolerance of embedded applications. Section 2 presents an overview of APEX while Section 3 discusses the quantitative objectives of APEX and Section 4 gives the details of the various tools and techniques that are provided by APEX. Finally, Section 5 summarizes the paper.

2. APEX – An Infrastructure for Software Reuse

A lot of work has been initiated recently on component-based software engineering, including middleware (CORBA), "design-by-contract," and certification of COTS software. ESC is developing the APEX (Advanced Programming Environment for Embedded Computer Systems) infrastructure to facilitate the use of COTS components and third party software. The APEX infrastructure includes automated code transformation and synthesis, automated qualification, and a framework for adapting a system to changes in its environment without sacrificing performance. APEX is sharply focused on embedded applications and DSPs, but the infrastructure encompasses a comprehensive, integrated solution that spans the entire product life-cycle

At the heart of APEX is the Online Repository for Embedded Software (ORES), a distributed collaborative web-based repository system connecting application developers with component vendors. A vendor's site offers support for component customization while a developer's site contains technical data regarding the components. The development process starts with a new COTS Aware Requirements Engineering (CARE) methodology to adapt a product concept to maximize the use of available COTS components in its implementation. A novel design method, DICE (Design for Independent Composition and Evaluation), is used to decompose the application into a set of independent subsystems and a framework for composing them together. Each subsystem is developed using the APEX AutoMAIC (Automated Modification And Integration of Components) utilities. These utilities interact with the vendor's web-site to customize components and mine the developer's ORES to generate glue code from existing design patterns and code templates. Each subsystem is then tested using a set of powerful simulation and analysis tools in EASiQA (Environment for Automated Simulation and Quality Analysis). EASiQA interacts with the vendor's web-site to generate instrumented versions of the components for performing coverage and other quality analysis. Once the subsystems have been independently validated, they are automatically composed together by the APEX Frameware, a compiler that generates a customized real-time operating system for the application based on its FSL (Framework Specification Language) specification. The final step in the development life-cycle is to apply the APEX High-assurance Automated Requirements and Design Enhancement (HARDEn) system to strengthen the reliability and security of the application.

The APEX infrastructure is unique in several ways.

- **Think COTS from the first step.** It is very difficult to try to achieve a good COTS-based design for an arbitrary set of requirements. Instead, APEX emphasizes the "front-end" aspects of software engineering to facilitate the effective use of COTS components and quality assurance during the later stages. An application-specific knowledge base of COTS components is being developed to guide the selection of functional and nonfunctional requirements that are amenable to implementation using the currently available suite of COTS components.
- **Think COTS throughout.** There has been some work on different aspects of COTS components, such as how to harden COTS components and how to assess the reliability of COTS components. The problem is that this type of focus makes it difficult to assure that any assumptions made will be satisfied in practice. For example, reliability assessment models usually assume that the COTS components have independent failure rates, but this may not be true in practice. Instead, APEX is focused on a narrow application area (embedded software) but encompass all aspects of the life-cycle process, including requirements specification and analysis, design, implementation, refinement, and evaluation. In this way, APEX ensures that, for example, the

assumptions made by an analysis technique will be guaranteed during the design and implementation steps.

- **Shift from a monolithic design to multiple, composable designs.** We are developing a method of decomposing embedded software systems into a set of Independently Developable End-user Assessable Logical (IDEAL) "micro-services" that can be automatically composed together to form the system. The micro-services will span both functional as well as nonfunctional aspects (repository, privacy, secrecy, authentication, etc.) and will be designed to insulate the application from changes in the set of available COTS components. The composition framework will be designed to guarantee that the properties of a system of micro-services can be inferred easily from the corresponding properties of the micro-services.

  Consider the design of a system corresponding to a requirements specification that defines user-visible features A, B, …, X. A monolithic s system consists of a number of components, P0, P1, P2, etc. These components are organized in a hierarchical structure. For example, component P0 may achieve its goal (specification) by using components P1, P2, P3, and P4. Similarly, P1 may be implemented using components a, b, and c. This type of a hierarchical structure or layered organization has several advantages. It enables bottom-up validation of the system and speeds up fault location. It also enhances system modularity since components can be encapsulated to hide their implementation details from the upper-level components.

  Such a hierarchical (or contractor-subcontractor) design structure is well suited to long-lived, well-understood, completely specified applications. However, it has limitations in handling rapidly changing applications. One reason is that it is very difficult to relate a component in a hierarchical system to specific user-visible features; hence, it is difficult to rapidly modify the software to adapt to changes in some feature, say feature B in the example discussed above. Similarly, it is difficult to assess the reliability of the system from the reliability of the components. Suppose the reliability of P0 is 1.0 and the reliability of P1, P2, P3, and P4 is 0.999. It is not possible to deduce the reliability of the system from this information. In fact, the overall reliability can be 0 (if P0 always triggers a defect in one of the lower-level components) to 1 (if P0 never triggers any defects in the lower-level components).

  To overcome these deficiencies of the hierarchical model, APEX incorporates a novel design methodology, DICE (Design for Independent Composition and Ervaluation), for decomposing embedded software systems into vertically-structured or orthogonal components. In this design, the specification is decomposed into separate pieces, A, B, …, X, such that (a) each piece can be developed independently and (b) each piece can be assessed independently at the end-user level. These two properties greatly enhance the customizability and quality of the system. Since each user requirement can be traced to a particular component, it is easy to identify the affected component and make changes to it. Similarly, since each component sees the same input distribution, it is possible to infer the reliability of the system from the reliability of its components.

- **Quantitative reliability assessment.** APEX emphasizes quantitative reliability assurance techniques. Models will be developed to strengthen the statistical reliability analysis by incorporating structural and functional information to achieve high confidence bounds. A framework will be developed to allow system-level properties to be inferred from component-level properties.

- **Application-specific focus.** APEX is a powerful, scalable technology for a specific but important application domain, namely, embedded telecommunications software for DSP-based platforms. This narrow focus enables us the special characteristics of DSP-based telecommunications software to be leveraged to develop deep knowledge bases, tools, and techniques for achieving accelerated development schedules and high quality assurance. The technology will address the special characteristics of DSP-based software, especially performance/real-time emphasis, low cost, small footprint solutions, heterogeneous dual/multi processor implementations, use of DSP Media accelerators, use of attached special purpose processors.

3. Quantitative Objectives of APEX

The goal of APEX is to achieve highly reliable and low cost embedded software systems without compromising performance. The quantitative expectations of APEX are to achieve 10-fold or more increase in productivity, 100-fold increase in customizability, 100-fold improvement in quality, high-assurance of critical requirements, and dynamically adaptable to changing environments and user requirements. These are discussed further in the following subsections.

3.1. 10-fold or more increase in productivity

This is being achieved by using tools and techniques to compress labor-intensive portions of the software lifecycle in order to narrow the gap between the output of domain experts (requirements specification) and the final software (code). A typical phase in a software lifecycle consists of requirements specification, design, implementation, qualification, and maintenance/operation. The most labor-intensive portions of this lifecycle are the implementation and qualification phases. APEX includes tools to reduce the implementation and qualification effort, thus resulting in accelerated product development. This compression of the software life-cycle also reduces the gap between domain experts, who draw up the requirements, and the final product. This allows domain experts to quickly evaluate the behavior of the implementation against their intentions and can lead to superior products due to fewer specification faults.

The APEX DICE methodology can reduce the implementation effort by enabling embedded applications to be decomposed into independently developable, end-user assessable logical (IDEAL) components and assembling each subsystem from COTS components and third party software. The use of COTS components carries with it the risk of failures due to the possibility of defects in the components and intentional or unintentional security breaches. Specialized automated code generation techniques (HARDEn) are used to enhance critical quality attributes of the resulting product, especially its reliability, security, and performance (Fig. 1). A framework is

being developed to allow the subsystems to be automatically composed together to obtain the system.

To assure high reliability, APEX includes methods of automating the qualification of the individual subsystems (Fig. 1) and an approach for composing the components together to guarantee certain key properties of the system. The underlying framework is designed to be simple and application-independent so that its reliability can be determined to a high degree of confidence and to ensure that it does not compromise the performance of the system.

3.2. 100-fold speedup in customization

This is achieved by ensuring that there is a one-to-one, end-user visible correspondence between the requirements and the IDEAL subsystems. Then, when a given piece of the specification changes, the affected subsystem can be rapidly identified, modified, and qualified. This approach also allows a software subsystem to be replaced by a hardware component. This capability can be used to develop products that have different price/performance characteristics, with a high-end version of the product being more hardware intensive than a lower cost version.

3.3. 100-fold improvement in quality

This is achieved by augmenting the framework to allow the reliability, resource requirements, and real-time performance of the embedded software to be determined from the corresponding properties of the IDEAL subsystems. Each IDEAL subsystem can have a much smaller state space than the entire application and, hence, its reliability can be assured to a higher degree of confidence. Also, techniques are being developed to harden COTS components used in the system to mask known as well as unknown defects in the components.

3.4. High assurance of critical requirements

This is achieved by augmenting the framework to ensure that defects in lower priority subsystems can never impact the functioning of more critical subsystems. The framework enables fault detection, isolation, and confinement. Since the behavior of each IDEAL subsystem is directly traceable to the requirements specification, it serves as a complete fault containment unit. Faulty components can be identified directly by analyzing the system output. Repair/recovery actions are also confined within each component separately.

3.5. Dynamically adaptable to changing environments and user requirements

This is achieved by enhancing the framework to allow an IDEAL component to be replaced or augmented by other components, including downloadable third party software. This also enables multi-paradigm implementations. For example, each IDEAL component can be implemented using the technology that is most suitable for it if the components can be composed dynamically. Some components can be implemented in software while others can be implemented in hardware.

4. APEX Tools and Methodologies

APEX includes an integrated set of tools and techniques to facilitate software reuse. These include ORES, CARE, DICE, AutoMAIC, EASiQA, and HARDEn. These are discussed briefly in the following subsections.

4.1. ORES

ORES contains the specification and categorization of COTS components and a web-based system for tracking information related to the components in the database, especially failure and dependency information that will be useful for assessing the reliability of the system. The information in the system is organized along multiple views. ORES provides advanced capabilities to help the programmer use the component. ORES also provides capabilities for acquiring and maintaining the information in the system.

4.2. CARE

This has two related aspects. One aspect of the methodology deals with tradeoff analysis between different nonfunctional requirements, such as non-recurring development costs, recurring development costs (customizability, maintainability), non-recurring usage (e.g., spatial) costs, recurring usage (e.g., performance, quality) costs. The second aspect incorporates a method of using the knowledge of the available components and their characteristics to guide the selection of functional requirements in order to ensure maximum usage of the COTS components, ensure good performance or reliability, etc.

4.3. DICE

Decomposing the requirements specification for a software system into separate views is a crucial step in simplifying the software and assuring high quality by making the specification more amenable to rigorous analysis. One of the earliest works was done by Zave [ZAV85]. The concept of multiple views has also been used in StateCharts which was developed by Harel in the mid-1980's [HAR87], Objectcharts which was developed by Coleman in the early 1990's [COL92], and other related methods. In the mid-1990's, Jackson applied this type of decomposition to an existing specification language, namely, Z [JAC95]. RSML, developed by Heimdahl and Leveson in the early 1990's [Hei96], is a significant extension to StateCharts with the goal of achieving more easily understandable and reviewable specifications. Decomposition methods that persist over the life-cycle include separation using rely-guarantee assertions (Lam, at UT-Austin [LAM94]), behavioral inheritance (Atkinson [ATK91]), and Aspect-Oriented Programming (Kiczales at Xerox PARC [KIC97]). There is substantial overlap between the philosophy of Aspect Oriented Programming and IDEAL components. However, there are also some important differences. For example, besides the emphasis on end-user assessability requirement for IDEAL components, another difference is that IDEAL components can be executed as separate processes in addition to being statically "woven" together to obtain one program.

The DICE methodology in APEX is focusing on identification of IDEAL subsystems and development of a suite of useful design patterns. DICE includes strategies for decomposing a given requirements specification into a set of IDEAL components that can be composed together to form the application. DICE includes a suite of useful domain-specific design patterns to speed up the design of embedded software systems.

4.4. AutoMAIC

AutoMAIC consists of a set of utilities for transforming components as well as for automatically generating the "glue" needed to compose software components together to implement a given subsystem. This step will also explore the use of code fragments and special purpose program generators in order to speed up the implementation process

Program transformation [ALM92] provides the means to statically compose the different components together. Transformation has also been used for improving software performance and portability and for transforming specifications into code, e.g., KIDS [SMI90] and MAPS. Transformation systems are becoming more and more sophisticated [MAT97].

4.5. EASiQA

EASiQA consists of two major tools. One is an automated system that generates application-specific test data generators for a given COTS component. The second tool is an environment simulator that allows individual subsystems or components to be tested independently. EASiQA also includes a set of tools and techniques for analyzing embedded software, especially the timing, security, and other properties. Several techniques have been developed for analyzing models of reactive systems, especially model checking [HOL97] and [BUR92] and partial-order methods [GOD96]. While these types of checks and proofs enhance the confidence in the correctness of the model, it is difficult to quantify the confidence level. Ebrahimi has developed a method of statistically evaluating software designs using the distribution of faults found separately by different reviewers [EBR97]. ESC is developing methods of achieving rigorous *quantitative* assessment of the reliability of the system. A variety of software reliability models have been developed over the past 25 years [LYU96], [MUS90]. One major impediment to the use of statistical software reliability models is the lack of rigorous techniques for deducing the reliability of a system from the reliability of its components. A key property of IDEAL components is that *every* component has the same state space, except for differences in goals and constraints; hence, every component sees the same operational profile. Hence, the IDEAL component has the important (and unique) property that the component reliability estimates can be statistically combined to obtain the system reliability [BAS99b,BAS99c].

4.6. Frameware

A problem with decomposition of a specification into simpler components is how to compose the components to obtain a system with assessable properties. One difficult problem is how to assure the consistency of the different views. Nonmonotonic logic, especially paraconsistent nonmonotonic logic, provides some support, but it cannot handle all types of inconsistencies. Finklestein [FIN94] and others have developed formal techniques to tag inconsistent specifications and remove them either manually or by using rule-based methods. This difficulty is compounded when different specification methods are used to specify different views, as proposed by Zave and Jackson [ZAV96].

The APEX Frameware allows micro-services to be implemented and evaluated independently and then be either composed dynamically or statically. Each component will be designed to be directly assessable at the end-user level and can be traced back to the requirements specification. This property facilitates fault-confinement, isolation, and reconfiguration. Methods will be developed to detect inconsistencies during relational composition and resolve them by assigning priorities to components.

The APEX Frameware provides one or more frameworks for composing the set of IDEAL subsystems together in order to obtain the final system. The Frameware ensures that the system properties can be inferred from the components properties. This includes the reliability and security of the system. It also includes methods of assuring that the real-time performance of the system can be inferred from those of its components. The Frameware also includes methods of identifying various environment conditions and adapting the system to operate optimally in a given environment. The conditions can range from device status, such as the energy remaining in the battery or the user settings, to network conditions, such as the available bandwidth or operating conditions.

4.7. HARTDEn

The HARDEn utilities provide a set of tools and techniques for enhancing the quality of the application. This includes wrappers to confine COTS components (security enhancement), wrappers to tolerate known and unknown faults in COTS components, and code analysis and optimization methods to improve the performance of the system.

Several approaches have been developed for handling failures dynamically after the software has been deployed, such as design diversity [AVI77], roll-forward recovery [PRA94], recovery blocks [RAN75], distributed and look-ahead executions [KIM89], data diversity [Amm87], defensive programming, exception handling, and forward recovery.

5. Summary

In this paper, we have presented an overview of an infrastructure for substantially increasing the likelihood of reusing existing software components in the development of highly reliable and customizable real-time embedded telecommunications applications. The APEX infrastructure encompasses a whole life-cycle approach, starting from the requirements engineering phase to design, implementation, quality assurance, enhancement, and operation. APEX consists of a collection of tools and techniques that access and mine code and related information from a collaborative web-based Online Repository for Embedded Software (ORES).

Information in ORES comes from the source code of the components, program templates and design patterns from existing applications, views and experience reports from field engineers, and operational reliability information from automated error logging and reporting utilities. Effort is now proceeding towards the development of interoperability standards for ORES as well as the tools and techniques in APEX.

6. Acknowledgment

This research was supported in part by the National Science Foundation under Award No. CCR-9900922 and by Alcatel and Texas Instruments.

7. References

[Amm87] P.E. Ammann and J.C. Knight, "Data diversity: An approach to software fault tolerance," *Proc. 17th Intl. Symp. on Fault-Tolerant Computing (FTCS-17)*, Pittsburgh, PA, June 1987, pp. 122-126.

[AlM92] T. Al-Marzooq and F.B. Bastani, "Program transformation in massively parallel systems," *Frontiers of Massively Parallel Computation*, McLeans, Virginia, Oct. 1992, pp. 498-501.

[Atk91] C. Atkinson, *Object-Oriented Reuse, Concurrency and Distribution*, Addison-Wesley, New York, NY, 1991.

[Avi77] A. Avizienis and L. Chen, "On the implementation of N-version programming for software fault tolerance during program execution," *Proc. COMPSAC'77*, 1977, pp. 149-155.

[BAS99a] F.B. Bastani, "Relational programs: An architecture for robust process-control programs," *Ann. of Software Engineering*, Vol. 7, 1999, pp. 5-24.

[BAS99b] F.B. Bastani, V. Reddy, P. Srigiriraju, and I.-L. Yen, "A relational program architecture for the Bay Area Rapid Transit System," *Conf. on High Integrity Systems*, Albuquerque, New Mexico, Nov. 1999.

[BAS99c] F.B. Bastani, V.L. Winter, and I-.L. Yen, "Dependability of relational safety-critical programs," *IEEE Intl. Symp. on Software Reliability Engineering* - Fast Abstract, Boca Raton, Florida, Nov. 1999.

[Bur92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, "Symbolic model checking: 10**20 states and beyond," *Info. and Comp.*, Vol. 98, No. 2, June 1992, pp. 142-170.

[Col92] D. Cooleman, F. Hayes, and S. Bear "Introducing Objectcharts or How to use Statecharts in object-oriented design," *IEEE Trans. on Softw. Eng.*, Vol. 18, No. 1, Jan. 1992, pp. 9-18.

[Ebr97] N.B. Ebrahimi, "On the statistical analysis of the number of errors remaining in a software design document after inspection," *IEEE Trans. on Softw. Eng.*, Vol. 23, No. 8. Aug. 1997, pp. 529-532.

[Fin94] A.C.W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh, "Inconsistency handling in multiperspective specifications," *IEEE Trans. on Softw. Eng.*, Vol. 20, No. 8, Aug. 1994, pp. 569-578.

[God96] P. Godefroid, D. Peled, and M. Staskauskas, "Using partial-order methods in the formal validation of industrial concurrent programs," *IEEE Trans. on Softw. Eng.*, Vol. 22, No. 7, July 1996, pp. 496-507.

[Har87] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. of Comput. Prog.*, Vol. 8, 1987, pp. 231-274.

[Hei96] M. Heimdahl and N.G. Leveson, "Completeness and consistency in hierarchical state-bed requirements," *IEEE Trans. Softw. Eng.*, Vol. 22, No. 6, June 1996, pp. 363-376.

[Hol97] G.J. Holzmann, "The model checker SPIN," *IEEE Trans. on Softw. Eng.*, Vol. 23, No. 5, May 1997, pp. 279-295.

[Jac95a] D. Jackson, "Structuring Z specifications with views," *ACM trans. Softw. Eng. and Meth.*, Vol. 4, No. 4, Oct. 1995, pp. 365-389.

[Kic97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.-M. Loigtier, J. Irwin, "Aspect-Oriented Programming," *Prof. European Cong. on Object-Oriented Programming (ECOOP)*, Finland, June 1997.

[Kim89] K.H. Kim and S.M. Yang, "Performance impacts of look-ahead execution in the conversation scheme," *IEEE Trans. Comp.*, Vol. C-38, No. 8, Aug. 1989, pp. 1188-1202.

[Lam94] S.S. Lam and A.U. Shankar, "A theory of interfaces and modules: I --- Composition Theorem," *IEEE Trans. on Softw. Eng.*, Vol. 20, No. 1, Jan. 1994, pp. 55-71.

[Lyu96] M. Lyu, (Ed.), *Handbook of Software Reliability Engineering*, McGraw-Hill and IEEE Comp. Soc. Press, 1996.

[Mat97] S. Matsuura, H. Kuruma, and S. Honiden, "EVA: A flexible programming method for evolving systems," *IEEE Trans. on Softw. Eng.*, Vol. 23, No. 5, May 1997, pp. 296-312.

[Mus90] J. D. Musa, A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Applications*, (professional edition), McGraw-Hill, 1990, pp. 178-180.

[Pra94] D.J. Pradhan and N.H. Vaidya, "Roll-forward and rollback recovery: Performance-reliability trade-off," *Intl. Symp. on Fault-Tolerant Comp. (FTCS-24)*, Austin, TX, June 1994, pp. 186-195.

[Ran75] B. Randell, "System structure for software fault tolerance," *IEEE Trans. on Softw. Eng.*, Vol. SE-1, June 1975, pp. 220-232.

[Smi90] D.R. Smith, "KIDS: A semiautomatic program development system," *IEEE Trans. on Softw. Eng.*, Vol. 16, No. 9, Sep. 1990, pp. 1024-1043.

[Zav85] P. Zave, "A distributed alternative to Finite-State-Machine specifications," *ACM Trans. on Prog. Lang. and Sys.*, Vol. 7, No. 1, Jan. 1985, pp. 10-36.

[Zav96] P. Zave and M. Jackson, "Where do operations come from? A multiparadigm specification technique," *IEEE Trans. on Softw. Eng.*, Vol. 22, No. 7, July 1996, pp. 508-528.