From *Proceedings of*
13^{th} *International Conference on*
Software Engineering, pp. 344-352
Austin, TX, May 13-16, 1991

Hardware/Software CoDesign: A Perspective

David W. Franke, Martin K. Purvis

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, TX 78759-6509

Abstract

Traditionally, computer systems development has been characterized as hardware engineers supplying general-purpose computing systems that are programmed by software engineers. However, developments in computing technology (e.g. computer aided design capabilities, ASICs, etc.) have brought about a reexamination of the traditional boundaries between hardware and software engineering and a need for a more flexible design style — one in which both the hardware and software design options can be considered together. Since current hardware and software design methodologies have their differences, a unified "codesign" approach must be developed that will comprise both the hardware and software points of view. Increasingly, computer system design of the future will require this codesign approach.

1 What Is CoDesign?

Until recently, computer systems development has been ordinarily characterized by the notion that hardware engineers supply general-purpose computing systems, which are then programmed by software engineers. Even as the use of microprocessors and firmware during the 1980's enabled more applications to be accomplished in software instead of hardware, this view of the computing world did not experience much change. Software engineering and hardware development were two activities which could be carried out relatively independently. Indeed, the U.S. Government military standard for computer system development [7] characterizes an independent hardware and software development. Thus software specialists could feel safe in the assumption that they needn't worry themselves too much about the "low-level" details of computer hardware. Furthermore, the growing acceptance of the "layered" model of computer systems architecture seemed likely to protect software practitioners from such concerns. But, now, developments in several areas of computing technology have brought about a reexamination of the traditional boundaries between hardware and software engineering and have cast doubt on the utility of maintaining a major boundary between the two. In this regard, we propose that a more fruitful approach to computer system design is to combine the hardware and software perspectives from the earliest stages of the design process and exploit the design flexibility and efficient allocation of function that such an approach offers. This is called "hardware/software codesign", or sometimes simply, "codesign".

1.1 Custom Chip Design

What are some of these developments that have created a need for the codesign point of view? One of them concerns improvements in computer-aided chip design capabilities. It is useful to consider this and other new technology developments in terms of the primary concerns that must be addressed when new computer systems are designed and produced:

- time-to-market
- performance
- system reliability

All three perspectives must be considered when assessing the effects of any new methods or technologies on future design practice, but for the moment we will concentrate on the first two items and consider the third one later (section 2.4). For example, in the realm of computing hardware, it has been customary to mark progress in terms of the miniaturization and functionality of conventional general-purpose computers. But now that 32-bit microprocessor-based personal computers can be placed on any desktop, further perfor-

mance enhancements along these lines are approaching a point of diminishing returns. Instead, focus is shifting to systems where the computational power is more distributed and specific to a given application. The reason for this shift is due in part to improvements in computer-aided design: custom-designed high-performance chips, such as graphics processors, can now be designed more easily and quickly, and consequently the time-to-market penalty for developing them has been reduced. As a result, trade-off considerations with respect to the three perspectives listed above now favor moving general-purpose computer architectures in the direction of distributed computing systems, with special-purpose computational elements devoted to specific tasks, such as graphics display, keyboard control, disk controllers, etc. Thus functionality is shifting from software (on the general-purpose computer) to special-purpose hardware.

1.2 ASICs

Even more significant to the shifting boundaries of hardware and software design has been the introduction of application-specific integrated circuits (ASICs) implemented in gate arrays, sea-of-gates technologies, gate matrix, and cell-based programmable logic arrays. These can be more specific to a given customer's application than the special-purpose but relatively generic chips mentioned above, and they too have been made possible by the current generation of automated CAD tools and chip fabrication capabilities. Using ASICs, specific algorithms of a computationally intensive nature can be implemented in silicon, enabling potentially large performance gains to be realized.

Note that a hardware design often realizes some degree of low-level parallelism, with the consequent speed-up that such parallelism entails. But often this parallelism is limited to some specific type of computation. For example, a microprocessor realizes parallel computations of 32 bits, but each package of 32 bits is treated sequentially, and so large-scale gains from such parallelism are limited. Because of this, efforts to develop general-purpose parallel computers have generally failed to live up to their expectations and hopes for them have met with disappointment. Not so disappointing, however, has been the fate of more special-purpose endeavors. Here the design has been tailored to the specific computation that the parallel realization "is good at". Image-processing, for instance, can take advantage of large-scale parallel implementations that have been designed for a specific set of computations. ASICs, therefore, often offer the opportunity of achieving parallelism for specific applications for which they have been custom-designed. In this way, "distributed parallelism" can gradually be introduced to computer applications on a case-by-case basis.

Shifting functionality from software to hardware, however, need not *automatically* entail a performance enhancement. The movement of certain processor functions *back* into software, for example, has enabled RISC processor technology to realize significant performance gains. So what needs to be emphasized is that the allocation of functionality into hardware and software is no longer as obvious and straightforward as it once was and that each application area requires a separate analysis concerning the optimal distribution of such functionality.

1.3 CoDesign

With improvements in computer-aided design making the boundary between hardware and software fuzzier and more problematic all the time, it is becoming clearly advantageous to have a more flexible design style – one in which both the hardware and software design options can be considered together. However, current hardware and software design methodologies have their differences, and careful thought must be given as to how to combine them. Admittedly, in both the ASIC and custom-chip styles of hardware design, an enabling feature has been the greater use of design abstraction (such as the use of standard cell libraries and behavioral level abstractions) in order to deal with increasing design sizes and orders of complexity. And software engineering too makes use of the notion of design abstraction – which means that both the hardware and software engineering disciplines have now developed tools and methods to deal with design abstraction hierarchies. Nevertheless, because of the traditional separation of the hardware and software development processes, the notational representation and modeling schemes that have been developed so far have been specific to the needs of either hardware or software engineers, and so it is not a simple matter to merge them. Here is the view of one designer:

> From other aspects, software design and silicon design exhibit almost a duality principle. If we take an abstract view of work performed inside computing systems, then some of the work is *communication* of data and some of the work is *computation* on the data. To oversimplify a bit, software design focuses on computation while VLSI design focuses on communication....Simply stated, software views processor cycles as the limiting factor and tries to make best use of the sequence of instruction cycles. VLSI design

> views communication bandwidth as the limiting factor and tries to make best use of the sequence of data operands. [16, p. 862]

In order to develop systems in the present age, where hardware and software trade-offs can be examined on a case-by-case basis, a new, unified approach is then needed that will comprise both the hardware and software points of view. The ideal way to go about developing a new system design should then be to start with an abstract notation such that each component or module is independent of its final realization in hardware or software. The partitioning between hardware and software, using such a codesign approach, could consequently be made in the most appropriate manner and not, as is the current customary practice, according to conventional wisdom. A recent textbook on software engineering makes note of this same idea:

> As the cost of special-purpose integrated circuits decreases and the speed of fabrication increases (for simple circuits, it is now possible to go from an idea to a delivered microchip in a few weeks), the borderline between hardware and software components is blurring. It is becoming increasingly cost-effective to delay decisions about which functions should be implemented in hardware and which functions should be software components. [30, p. 182-183]

> What is important in the design process is to delay hardware/software partitioning as late as possible in the design process, and this implies that the system architecture must be made up of stand-alone components which can be implemented in either hardware or software. [30, p. 182-183]

Inasmuch as codesign systems are usually of a complex nature, they can be difficult and time-consuming to design. Further, as discussed in [11], the majority of product production costs are determined during design. And clearly the time consumed for the design of such systems can be even more critical: a widely quoted study ([25], cited in [4]) states that a six month delay in product release was more damaging to profitability than a 50 per cent cost overrun during the product design cycle.

However progress in the codesign area is only in the initial stages, particularly with respect to upstream approaches, and there is much to be done in order to realize the full benefits from this approach.

1.4 CoDesign and System Reliability

So far we have made little mention of system reliability, the third category important to design. This is a consideration whose importance grows with the increasing complexity of computer system designs:

> As computer applications become more diverse and pervade almost every area of everyday life, it is becoming more and more apparent that the most important dynamic characteristic of computer software is that it should be reliable. [30, p. 16]

Computer system professionals are increasingly involved these days with the development of "reactive" systems [15, 24], which continually interact with humans and their environment. Typical examples include telephones, automobiles, communication networks, computer operating systems, avionics systems, etc [14, p. 2]. These systems comprise components from

- hardware
- software
- users and objects from the real world

In order to design and understand such systems, it is necessary to model all of three components in an integrated and consistent manner:

> Progress in developing software systems requires a fundamental appreciation that those systems are more than just a collection of parts and that software is embedded in larger systems with a variety of physical components: design of such systems must deal with both of these issues. Design of software systems must also take into account the fact that the whole system includes people as well as hardware, software, and a wide variety of material elements.[6, p. 283]

In order to develop robust and reliable complex systems, it will be necessary, as is the case in all engineering and scientific disciplines, to develop and apply formal mathematical methods wherever possible. Progress has been made in this area, but much remains to be done. In many respects practical advances in this area will be delayed until improved modeling and specification methods are available. Atsushi Takahara from NTT, Japan, commenting on formal verification techniques, observed:

> In higher level systems, we will have to think much more about the specification method than

> the theoretical techniques if such techniques are to be useful. The most pressing problem is how to express what we want to make. In fact, that *is* the problem. And in all the formal verification systems we've discussed today, everybody has assumed that we have a correct specification for verification. We should think much more about how to construct a language to express that specification, including the semantics and the syntax. [32]

While the goal of attaining provably correct computing systems may be extremely difficult to attain, serious efforts in the direction of greater computer system reliability, particularly for systems critical to human life and property, must be made on systems currently being designed. In the end, it will be necessary to provide a more rigorous and unified approach to system modeling and specification, covering all three of the above listed system components. However, advances can only be expected to take place incrementally, and the place to begin this process is one where significant gains can be realized in the near-term: merging the hardware and software modeling domains. For this reason advances in the area of hardware/software codesign can be expected to lie along the path towards greater complex system reliability.

1.5 Concurrent Engineering and System Engineering

The goal of providing a more uniform framework for the design of systems has a general appeal, of course, and the idea has been presented in a number of quarters. For example, efforts to come to grips with the problems associated with the increasingly complex nature of product design has led to attempts to bring about a tighter integration of its various phases, known as "concurrent engineering". One characterization of concurrent engineering (CE) asserts that it

> ...promotes a freer interchange of information between multiple engineering disciplines such as testing, manufacturing, reliability, maintainability, and all other disciplines that can contribute to making a better product. [11]

The accepted formal definition of CE defines it as

> A systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. [34]

Now with such a general prescription, CE can be thought to encompass almost all aspects of systems design, including hardware/software codesign. However, most references to concurrent engineering to date have centered on considering traditionally serial issues (e.g. design, test, manufacture) concurrently. Codesign, as we are outlining it here, attempts to better integrate two concurrent activities, namely the design of the hardware and software components of a system. Consequently, we view codesign as one aspect of future concurrent engineering capabilities and environments.

1.6 Who are the Codesigners?

What kinds of systems involve hardware/software codesign? We've already mentioned some example "reactive" systems. These are complex systems with hardware and software components that are constantly in execution, that is, reacting to their environment. The Boeing 767 airplane, for example, can be viewed in these terms. Although one's immediate picture of a 767 might be that of a large piece of hardware, one should also be aware that there are also about 5 million lines of software code [21] associated with that "system". This simply reflects the fact that to an increasing degree, many engineered systems, from automobiles to video cameras to aircraft, contain an integrated mix of software and hardware. Some other categories to consider are

- embedded systems
- transaction systems
- systems with high performance requirements
- "smart" I/O components, such as image processors, speech recognition components, etc.
- special-purpose computational elements, such as those involving neural networks and analog devices

In fact, many of the systems to be designed in the coming years will have the character of codesigns. As Elliott Organick observed,

> Microelectronics technology has advanced so rapidly and been so successful that we are now having to build large systems with a multitude of diverse, interacting components. Some components of these systems exhibit distinct architectures and may, in fact, be implemented following different choices of data abstraction realized in a variety of logic and circuit technologies. When we as designers understand how to build such systems, we are no longer *just* software engineers of *just* hardware engineers – we become

> "heterosystems" engineers, a more accomplished breed of engineering professional concerned with building systems that are truly heterogeneous in the fullest sense. [23, p. 31]

And Alfred Hartmann echoes this sentiment:

> The shifting boundaries of suitability between the two implementation alternatives [hardware and software] may require competent systems designers to be "bilingual", conversant in both design styles as the application warrants. [16, p. 873]

It is our contention that codesign is somewhere in the future of everyone involved with computer system design. In the following sections specific issues with respect to codesign are discussed in more detail.

2 Issues in CoDesign

Current system development methods [2, 7] specify series of steps from system requirements capture to final integration and test. While these steps may be iteratively encountered (i.e. design activity may regress to previous steps in the sequence), they do not capture the concurrent character of many real design activities. An alternative model of the design process is one in which the refinement of requirements, specification, design, implementation, and verification occurs concurrently, with progress made opportunistically or via a least commitment or delayed commitment [33] strategy (see Figure 1). Such designs often appear to have followed a serial design method upon completion by producing the appropriate, consistent documents for the individual design steps.

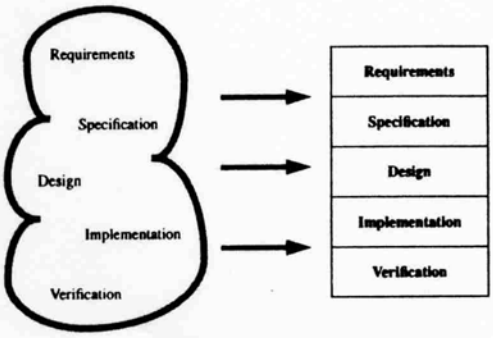

Figure 1: Concurrent Design Model

Systems whose implementation will be expressed in hardware and software exacerbate the problem of a serial development method in that the decisions made in the hardware or software design activity can have significant impact upon or reliance upon decisions in the complementary domain (software or hardware, respectively). This situation is best understood in examining the system development method given in [7] and shown in Figure 2.

Design issues and decisions that can cause the development process to regress to a previous step in one domain may also require a similar regression in the complementary domain. The system engineering activity has been created to address exactly such issues. However, the analysis and models created during system engineering are not carried through the design process and refined as the design progresses. Additionally, improvements in hardware implementation technology and design tools and methods have changed the economics of what should and can be implemented as hardware in a cost effective manner. Hence, the task of the system engineer is becoming more difficult, and requires better means for evaluating alternative architectures and allocations of function into hardware and software with respect to product performance, non-recurring (development) costs, recurring (manufacturing) costs, test, reliability, maintenance, and evolution.

Specific issues in codesign are presented in subsequent sections. Some of these issue are generic to the design process, but are addressed in the codesign context as we feel they are currently unaddressed in a generic context and hence for codesign in particular.

2.1 Requirements

The issue of requirements capture and elaboration is important for all design domains including hardware/software codesign. Requirements elaboration is an excellent example of the utility of the concurrent design model. Some aspects of a product design require that some non-trivial effort in specification, design, or even implementation be expended in order to understand the requirements. The user-interface (hardware or software) of a product often requires some form of prototype that can be examined and experimented with before precise requirements can be formulated for that interface [3]. The plausibility-driven design method [17] recognizes the need for elaborating requirements during specifications and design, as opposed to a strictly serial approach (requirements capture, specification, design). The point to be made here is not that it is impossible to capture all require-

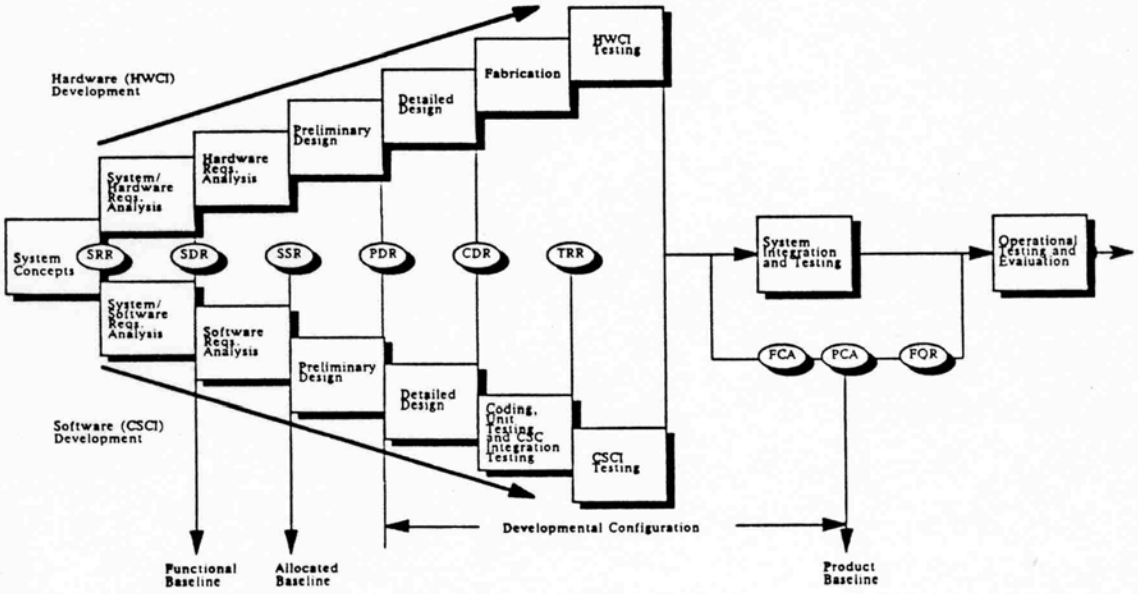

Figure 2: System Development Method, DoD-STD-2167

ments before starting specification or design efforts, but that such capture is not always possible and design methods should better support such design cases. In fact, capturing all product requirements as the initial stage of product development is preferred in that it restricts the space of possible designs for the product and hence makes specification, design, implementation, and verification efforts more straightforward.

In addition to the capture and elaboration of requirements, it is important to integrate the consideration of these requirements into subsequent activities in the development process, particularly when verifying that requirements have been met. From the point of view of codesign, conformance to product requirements will necessarily require the specification, design, or implementation to be evaluated with respect to realization in and trade-offs between hardware and software. This theme of integration of product information occurs throughout the individual issues cited in this section, and hence is summarized as the final issue.

2.2 Model Continuity

An important tool used in system engineering and high level architectural exploration is the development and evaluation of models. This evaluation sometimes occurs as a thought or paper experiment, but more frequently is occurring as formal analysis and computer simulation of the models. Such models provide a design expression that can be input to estimators (e.g. development or manufacturing costs) and to simulators that most often examine performance issues. However, these models and the associated system level analysis are rarely carried forward through the design process.

Upon interviewing designers, we have noticed that two specific problems are commonly expressed about modeling. First, in some cases it can be as time consuming to develop a meaningful model as it is to design and implement a prototype. Analysis of the prototype can often give more accurate information on cost, performance, and other aspects of the design. Second, in cases where models have been developed and analyzed, they are often discarded once the analysis is completed and are not revisited in the remainder of the development process. One reason for this is that maintaining the model to conform with subsequent architectural and design decisions requires additional effort with apparently little or no return. We feel that the first issue, namely the difficulty of developing architectural or high level models, is beginning to be addressed by various research centers in industry and academia [5, 12, 13, 19, 26, 27, 28, 29]. However, the second issue, namely maintaining conformance of such models and utilizing them throughout the development process, is not currently being addressed by commercial offerings or in the research community. In particular, the connection between system level models, designs, and implementations are not maintained to the degree

needed to support 1) projection of high level architectural rationale into detailed design decision making and 2) reflection of detailed cost, performance, reliability, test, and maintenance information back into higher level (architectural) descriptions. We call this the model continuity problem, and it is another instance of the need to integrate product information throughout the development process.

The projection of system level analysis throughout product development is relevant even in those design activities in which function allocation between hardware and software is straightforward. In this case, it is still important and beneficial to evaluate trade-offs in one domain or the other (i.e. the hardware or the software domain) with respect to the total system design, considering the consequences for both hardware and software. It is sometimes the case in joint hardware and software development projects that significant software design activity is delayed until such time as a working hardware prototype is available [18, 20]. In such cases, the software design and implementation often becomes the critical path to product introduction. If models of the hardware design can be constructed and then executed in conjunction with software, the software design and implementation can begin earlier in the development cycle, thereby shortening the product's time-to-market.

2.3 Metrics

The purpose of developing system models is to evaluate these models with respect to product requirements and goals, such as performance, cost, and reliability. It is important to have metrics 1) to express these product requirements and goals and 2) to measure and compare design alternatives. Identification and application of metrics is currently a problem independently in the hardware and software design domains, as stated in [8]:

> As a design evolves in a top-down approach, a direct relationship in terms of parameters and metrics between the hierarchical levels typically is not generated. The formal data consists only of block and flow diagrams and a set of requirements. Modeling and analyses rarely cross the hierarchical levels. ... Another aspect of this problem is that there are few metrics established to relate multiple levels of a design. Many characteristics at the high levels lack meaningful or complete metrics. For example, few metrics exist for reliability at the conceptual level, and no general method exists for relating reliability characteristics between hierarchical levels.

For system models that will eventually be realized in a combination of hardware and software, these metrics must be relevant to both hardware and software realizations of system functions. Since system level models are abstractions of implementation, estimators for the metrics of interest are required to evaluate system models.

In addition to metrics about the product, it is interesting to develop metrics for designed products (e.g. product quality) and for the design process itself. Metrics of the design process allow the contribution of codesign tools to be evaluated. For example, it is valuable to measure the impact of tools and design methods on the productivity of a product development organization.

2.4 Requirements Verification

The discussion of codesign issues opened with requirements capture and elaboration, and here we return to the topic of requirements from the perspective of verifying that an architecture or design meets these requirements. Requirements verification techniques range from simple observation to simulation to formal proofs of conformance. As more rigorous verification methods are employed, the associated requirements necessarily become more formal. Formal expression of requirements is intimately related to the selection of the metrics in which the requirements are expressed. Clearly the expression of requirements, the expression and evaluation of models and designs, and metrics are critical to achieve requirements verification. Here again, the integration of product information is needed.

2.5 Product Data Model

The issues described to this point are interrelated in one aspect or another, and highlight the need for the collection of and access to information about the development of a product. This database of product information can support product related activities throughout the life-cycle of the product, and can also contribute to the development activities of other products. For example, a product database can provide information on design and design rationale so that aspects of a product design can be reused in subsequent design activities.

To understand the information requirements of such a product database, it is important to develop a data model that expresses the logical content of the database and the relationships between items in the database. Such a data model can also be used in an

operational sense, providing a data abstraction whose implementation may be changed without impacting applications that access the database, and allowing the automatic generation of associated interface or access routines.

3 A Look to the Future

> A system-level design methodology will be an important enabling factor in producing the electronic product designs of the future. Equally important, CAD tools and services that support the system-level design methodology must be produced. ... Fourth-generation CAD systems will differ from today's tools in that they will support system design, including hardware, software, and packaging. [31]

> In the design environment of the year 2000, separate hardware and software islands will no longer exist. Applications of all sorts will be bonded together by a "software glue," which will unite diverse areas of interest and intertwine them inextricably. The process of design itself will be embodied in the system. And today's software engineer will become tomorrow's system designer. [1]

Commercial offerings and some university research to date [10, 12, 13, 19, 22, 26, 27, 28, 29] have primarily focused on the evaluation of system performance across alternate hardware/software configurations. Certainly performance is one important consideration when evaluating function allocation trade-offs between hardware and software, and these systems represent the initial steps toward codesign support. However, in the broad context of codesign, where *system reliability* and *time-to-market* are viewed together with *performance*, a larger theme emerges: that of of integrating information across the various stages of the product life-cycle and across the domains of hardware and software. The long term image of this theme is a "unified product model" that is realized in an electronic form accessible to tools and interested parties involved with a product throughout the product life-cycle. In addition to information about products, knowledge about the processes of product conception, development, manufacture, and support as well as knowledge about the organizations which execute these processes can be represented and utilized in the endeavor of "enterprise engineering". Doug Englebart [9] characterizes such integrated, electronic representation of information, associated tools for interacting with this information, and collaboration capabilities among widely distributed knowledge workers as key to "boosting the knowledge work of organizations".

Given such a base of knowledge and information, appropriate tools, environments, and interfaces for accessing and examining this information will leverage most all of the activities undertaken by an organization, from top management levels to individual contributors.

The investigation of codesign research issues is a next logical step toward these longer term goals, building on previous and current research in the areas of computer aided engineering in the design domains of software and hardware.

References

[1] Laszlo A. Belady, in *MCC Software Technology Program* (brochure), 1990.

[2] Barry W. Boehm, "A Spiral Model of Software Development and Enhancement", in *Computer*, Vol. 21, No. 5 (May 1988), pp. 61-72.

[3] Frederick P. Brooks, Jr., "No Silver Bullet: Essence and Accidents of Software Engineering", in *IEEE Computer*, Vol. 20, No. 4 (April 1987), pp. 10-19.

[4] Ralph K. Cavin, III, Jeffery L. Hilbert, "Design of Integrated Circuits: Directions and Challenges", in *Proceedings of the IEEE*, Vol. 78, No. 2 (February 1990), pp. 418-435.

[5] K. M. Chandy, J. Misra, *Parallel Program Design*, Reading, Massachusetts: Addison-Wesley, 1989.

[6] Computer Science and Technology Board, "Scaling Up: A Research Agenda for Software Engineering", in *Communications of the ACM*, Vol. 33, No. 3 (March 1990), pp. 281-293.

[7] DoD-STD-2167

[8] "Evaluating Design Process Problems", in *Electronic Engineering Times*, pp. 34-38, July 16, 1990

[9] Douglas C. Englebart, "Bootstrapping Organizations into the 21^{st} Century", SIGCHI presentation, May 15, 1990, Austin, TX.

[10] Gerald Estrin, Robert S. Fenchel, Rami R. Razouk, Mary K. Vernon, "SARA (System ARchitects Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Sys-

tems", in *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2 (February 1986), pp. 293-311.

[11] Stephan Evanczuk, "Concurrent Engineering: The New Look of Design", in *High Performance Systems*, April 1990.

[12] *Foresight Product Description*, Athena Systems Incorporated, March 1989.

[13] Geoffrey A. Frank, Deborah L. Franke, William F. Ingogly, "An Architecture Design and Assessment System", in *VLSI Design*, August 1985, pp. 30-38.

[14] David Harel, "Statecharts: A Visual Approach to Complex Systems", MCC Technical Report STP-107-86a, February 1986.

[15] David Harel, "On Visual Formalisms", in *Communications of the ACM*, Vol. 31, No. 5 (May 1988), pp. 514-530.

[16] Alfred C. Hartmann, "Software or Silicon? The Designer's Option", in *Proceedings of the IEEE*, Vol. 74, No. 6 (June 1986), pp. 861-874.

[17] Alan R. Hooton, Ulises Agüero, Subrata Dasgupta, "An Exercise in Plausibility-Driven Design", in *Computer*, Vol. 21, No. 7 (July 1988), pp. 21-31.

[18] Shankerappa Hulsoor, personal communication.

[19] *Integrated Design Automation System IDAS Product Description Summary*, JRS Research Laboratories, Inc., June 1988.

[20] Andrew Kern, Al Blazevicius "A Concurrent Hardware and Software Design Environment", in *VLSI Systems Design*, August 1988, pp. 34-40.

[21] Geoffrey McIntyre, FAA, "MacNeil/Lehrer News Hour" (interview) 21 February 1990.

[22] Richard Nass, "Develop Hardware and Software Simultaneously in One Environment", in *Electronic Design*, October 26, 1989, p. 137.

[23] E. L. Organick, T. M. Carter, M. P. Maloney, A. Davis, A. B. Hayes, D. Klass, G. Lindstrom, B. E. Nelson, K. F. Smith, "Transforming an Ada Program Unit to Silicon and Verifying Its Behavior in an Ada Environment: A First Experiment", in *IEEE Software*, Vol. 1, No. 1 (January 1984), pp. 31-49.

[24] A. Pnueli, "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends", in *Current Trends in Concurrency*, J. W. de Bakker et al. (eds), Lecture Notes in Computer Science, Vol. 224, Springer-Verlag, New York, 1986.

[25] D. G. Reinersten, "Whodunit? The Search for the New-Product Killers", McKinsey & Company, New York, July 1983.

[26] Gruia-Catalin Roman, Mishell J. Stucki, William E. Ball, Will D. Gillett, "A Total System Design Framework", in *IEEE Computer*, Vol. 15, No. 5 (May 1984), pp. 15-26.

[27] "SES/*workbench*^{TM}: A Multilevel Design Environment for Modeling and Evaluation of Complex Systems", Scientific and Engineering Software, Inc., May 1989.

[28] V. Shen, C. Richter, M. Graf, J. Brumfield, "VERDI: A Visual Environment for Designing Distributed Systems", *J. Parallel Distrib. Comp.*, 9 (1990), pp. 128-137.

[29] Connie U. Smith, John L. Cuadrado, Geoffrey A. Frank, "An Architecture Design and Assessment System for Software/Hardware Codesign", in *Proceedings of the* 22^{nd} *Design Automation Conference*, Las Vegas, June 1985, pp. 417-424.

[30] Ian Sommerville, *Software Engineering*, Third Edition, Addison-Wesley, Workingham, England.

[31] Jerry Sullivan, "Toward Fourth-Generation Design Automation Tools", in *Electronic Engineering Times*, January 29, 1990.

[32] Atsushi Takahara, "A D&T Roundtable - Formal Verification: Is It Practical for Real-World Design?" Carl Pixley (moderator), in *IEEE Design & Test of Computers*, Vol. 6, No. 6 (December 1989), pp. 50-58.

[33] Harold Thimbleby, "Delaying Commitment", in *IEEE Software*, Vol. 5, No. 3 (May 1988), pp. 78-86.

[34] Robert I. Winner, et al., "The Role of Concurrent Engineering in Weapons System Acquisition", IDA Report R-338, IDA, Alexandria, VA. Also Defense Technical Information Center report AD-A203-615.