HARDWARE PROJECT MANAGEMENT - WHAT WE CAN LEARN FROM THE SOFTWARE DEVELOPMENT PROCESS FOR HARDWARE DESIGN?

Rolf Drechsler, Andreas Breiter
Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{*drechsle,abreiter*}*@informatik.uni-bremen.de*

Keywords: hardware design, description language, project management, software engineering, project

Abstract: Nowadays hardware development process is more and more software oriented. Hardware description languages (HDLs), like VHDL or Verilog, are used to describe the hardware on the register-transfer level (RTL) or on even higher levels of abstraction. Considering ASICs of more than 10 million gates and a HDL to gate ratio of approximately 1:10 to 1:100, i.e. one line of HDL code on the RTL corresponds to 10 to 100 gates in the netlist, the HDL description consists of several hundred thousand lines of code.
While classical hardware design focuses purely on the development of efficient tools to support the designer, in industrial work processes the development cycle becomes more and more important. In this paper we discuss an approach, where known concepts from software engineering and project management are studied and transferred to the hardware domain. Several aspects are pointed out that should ensure high quality designs and by this the paper presents a way working towards a more robust design process by a tight integration of hardware design and project management. The intention of this work is not to provide an exhaustive discussion, but many points are addressed that with increasing circuit complexity will become more and more important for successful ASIC design.

1 INTRODUCTION

Nowadays application-specific integrated circuits (ASICs) consist of several million gates. To handle this complexity a high level of abstraction is needed as provided by hardware description languages (HDLs), like VHDL or Verilog. Efficient tools exist to guarantee the translation of HDLs to netlists that can be automatically placed and routed. Even though in most cases experienced designers have to optimize manually the description to result in high performance circuits, the tool support is very satisfying. Several methods for efficient high-level and logic synthesis have been developed in the past few years that are well understood (for an overview see (DeMicheli, 1994; Hachtel and Somenzi, 1996)). Due to changes in the technology, these techniques are steadily improved, e.g. by also considering layout aspects during the synthesis step, but the overall flow of the design process remained the same. In contrast the form of the description changed significant over the past years. While ASICs described in HDLs contained only a few thousand lines some years ago, today's design can consist of up to several hundred thousand lines of code. By this observation it is only a small step to ask whether the concepts that evolved over the last couple of years (and are still applied in industrial practice) are adequate to cope with the emerging complexity of ASICs to be developed in the near future. The same problem - in a different formulation - arose in the context of software development 30 years ago. Starting from very simple programs huge software systems grew. Detailed studies showed the high demand for a structured development process. The important parameters to guarantee high quality designs were determined and evaluated resulting in the field of *software engineering* (Naur and Randell, 1969; Sommerville, 2004). Moreover, the cost of software projects increased, making it necessary to develop adequate cost estimation models. In this paper we will study the concepts proposed for the software development process and point out that most of

the aspects also apply to hardware design based on HDLs. We will discuss important aspects, like specification, (interface) design, and documentation. The analysis leads to the formulation of a concept that covers the complete hardware development process and can easily be integrated in existing flows. This results from the observation that several of the steps described in the following are quite intuitive and some might already be integrated in the development cycle, but in most cases without considering the overall structure, e.g. some well accepted rules are used as coding guidelines. As a consequence of the discussion it follows that modern hardware design can and should consider and incorporate techniques known from software engineering and project management to allow high quality designs. Analogously to the software domain, the resulting process describes the tight integration of project management and hardware design. Note, that the points addressed in the following are not complete and the paper can only be seen as first step in this direction. Starting with the analysis of similarities between hardware and software development processes, we will introduce a process model for hardware-engineering. The three constituting layers will be explained in detail, focusing especially on project management and aspects of quality management within the hardware development process. The paper closes by reflecting core problems in hardware design which might occur during the development process.

2 SIMILARITY BETWEEN HARDWARE AND SOFTWARE DEVELOPMENT PROCESS

The "classical" hardware development process - in a simplified way - is shown in Figure 1. Starting from an initial idea, a specification is written as a text. This is later formalized, such that it can be simulated, e.g. in a programming language like C. Then a designer writes the code in a hardware description language, like VHDL or Verilog. In a next step the netlist is generated by automatic logic synthesis tools and mapped to the layout later. From this description the final chip is produced. It can be observed that the main part has to do with coding, similar to software development.

Recent years have seen a pronounced shift in software engineering practice away from linear waterfall process models toward the iterative approaches pioneered by Boehm (Boehm, 1988) and others. Boehm's spiral model point out the basic phases of software development: inception - analysis - risk assessment - design - implementation - test - deployment - maintenance. Modern process models tend to cycle through basic phases (iterative model) and to use at each development step all information available from artefacts created in the process. Additionally, we can find further similarities between hardware and software description languages. For hardware and software high level languages have been defined to simplify the development process. For software, the goal was to leave the low level descriptions, like assembler, while on the hardware level the corresponding format was the schematic entry. Starting from small descriptions, in the meantime complex systems - hardware and software - can have up to several thousand lines of code. While in the early beginning single developers build a system, today up to hundred designers work on a project in parallel. Even though in first programming languages for hardware were very similar to software descriptions (see e.g. (Chu, 1965)), aspects of concurrency have been added to HDLs in the meantime. But successful concepts from software are also integrated in HDLs, like e.g. object orientation.

3 HARDWARE-ENGINEERING: MANAGING THE HARDWARE DEVELOPMENT PROCESS

Taking this into account, it becomes obvious that many problems occurring in hardware development are very similar to what has been considered in software several years ago. In this section, different topics are considered and their importance to high quality hardware development is discussed. Here, it should be noted that several recent studies of larger hardware design projects showed that many of the design bugs that are found during the verification step do not result from "real" design errors, but from an incompletely specified design flow (see e.g. (Krishnamurthy et al., 2001; Bentley, 2001)). We will separate the process in three interconnected layers (see Figure 2) which will then be described in more detail.

3.1 Product Development Cycle

To get a better understanding of the project workflows, the development process is subdivided in three (more or less) independent steps which are identical with the "classical" management cycle from planning to organizing and to controlling.

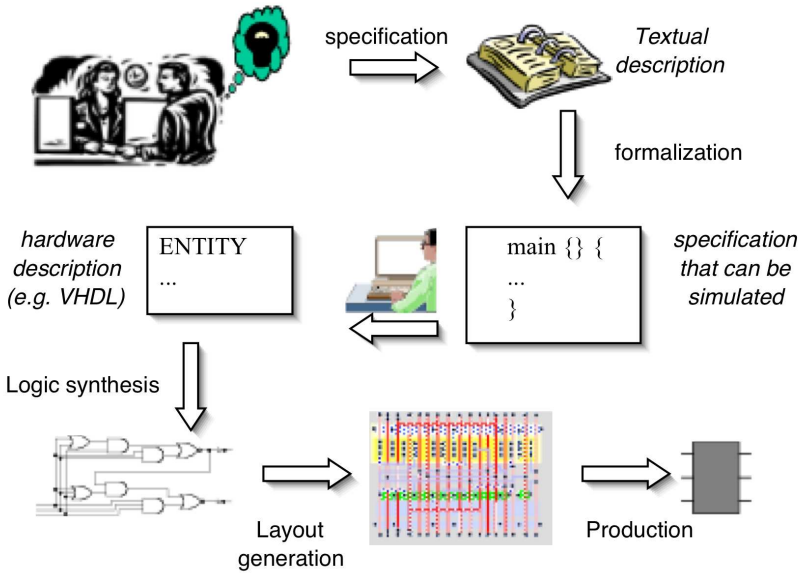

Figure 1: Hardware Development Process

Planning: This phase includes the description of the hardware architecture and the resulting sub-components. Additionally, general aspects of availability of human and technical resources, financing, costs etc. as should be integrated. Especially, already at this stage methods of quality assurance and prevention, i.e. what happens in an "emergency", should be discussed to allow a risk management. During the planning phase it is also important to describe the responsibilities of each team member with in the production process.

Production: Assuming that all CAD tools are selected, sub-tasks are assigned and the design and coding phase starts. Already during the creation, the individual modules are verified by simulation or formal techniques. From the experience of several projects of large size, it can be concluded that it does not make much sense to start with the complete team at once. It is better to first take a few - but highly qualified - designers. These first clarify the technical flow, since in some cases it can be necessary to go back to the planning phase, if problems occur. Usually, one can distinguish between the *hardest-first strategy* and the *easiest-first strategy* depending on the goals and the schedule of the project.

Checking: This process does not start after the product has been build, but has to be considered during the complete project cycle. (Of course, at the end the overall functionality is checked.) This verification process already starts at the module level, but several test, like consistency of interfaces, can only be considered after all sub-components are available. Still, most of nowadays ASICs are verified by simulation, while formal verification (equivalence checking, property checking) or semi-formal verification (symbolic simulation, assertion checking) become more and more important. But the checking phase also involves evaluation of the complete project. It has to consider the instruction and motivation of the team, decision making, adaptation of goals (if needed), validation, product delivery, and the final evaluation of the project.

Dependent on the phases a high interaction might be necessary. From the initial project planning to the final product delivery it usually takes more than three years for multi-million gate ASICs. Thus, the requirements can change during this time and the organization structure of a project must be flexible enough to take these "re-definitions" into account.

3.2 Hardware Development Process

The hardware development process can be divided into several sub-phases which are quite similar to those in software engineering. In the following the project phases are addressed, not the design phases. The design steps as described in e.g. (DeMicheli, 1994) remain the essential part of the design phase.

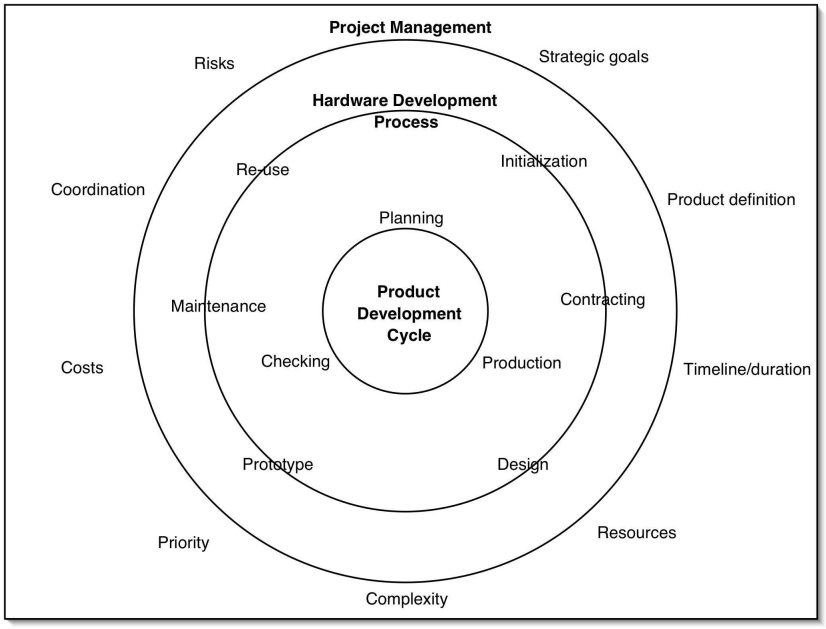

Figure 2: Three Layer Model of Hardware Engineering

But to run a successful project, it is important to consider and understand the overall workflows.

Initialization: Beside getting to know the project partner, in the very early phase of the project it has to be clarified, whether everyone involved has enough knowledge about the hardware to build. Already in this phase all "constraints" of the project (resources, available know-how, time, etc.) should be considered and written down in the form of a protocol. It is important in this phase to be as realistic as possible and not to be to euphoric about the chance of a new project.

Offer/contract: Based on the first contact an offer should be made and if accepted the contracts can be signed. The offer must be as specific as possible and should cover a detailed project description, dates, structure, warranty etc.

Design: The design phase has been studied intensively over the past twenty years and in the meantime is well understood. Several commercially available tools support the designer. From the project management side, it is important to first concentrate on the main features of the hardware to be build, while some special "extensions" can often be added in a later stage.

Prototype: For larger projects it is often helpful to first build a prototype. Depending on the type of hardware, the prototype can later be extended to a complete project. The role of the prototype should be determined first, and then the implementation should start. In the software domain a classification in horizontal and vertical prototypes is used which cannot be transferred directly to the hardware domain. But the way to build the prototypes, i.e. exploratory prototyping, experimental prototyping and evolutionary prototyping (see e.g. (Floyd, 1984; Vonk, 1990)), gives helpful insights. Most prototyping approaches draw the attention to the end user as the key actor in the adoption process. The development of a prototype is also often used to evaluate new CAD tools and to try to integrate them in an existing workflow. This becomes especially important, if new design processes are used.

Maintenance: The IEEE defines software maintenance as "the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment." (IEEE, 1993). It can either be perfective (increasing performance, maintainability, efficiency), adaptive (changing software to match different requirements or processing environments) or corrective (identifying and correcting failures in performance and/or implementation) (Lientz and Swanson, 1980). In hardware development, while the component is in use - we ideally assume that no redesign was necessary - the requirements may change. Nowadays hardware is in use only for about two to three years on average. But often slight modifica-

tions are sufficient to extend the product to satisfy the new demands. Here it is very important to have already considered reuse aspects in the design phase (see e.g. (Keating and Bricaud, 1999)).

Re-use: If we take reuse into account, many components are in use for 10 years (or even longer). For this, the interface design and conforming with standards is becoming a key issue. This should already be considered during the planning phase (see above).

The decisions in the early project phases influence many individual steps that become very important in the later stages. If they are not taken into account at the beginning, it can become difficult - or even impractical - to run a successful overall project.

3.3 Hardware Project Management

The complete design process - from the initial planning phase until the final delivery of the product - can be regarded as a project. The general tasks of project management are shown in Figure 3.

From the description above, it easily follows that the project organization and management has crucial impact on the overall success of the hardware design. Relating this to the previous section, there are different roles within project teams: management (planning and checking, but not production), development (planning (in the sense of working within the defined goals, meeting the deadline, staying within the planned resources) and production, but not checking[1]), and monitoring and controlling (checking and production, but not planning). These roles have to be clearly assigned to each team member prior to the project start. Otherwise, the competence and - in case of problems - the responsibility are not clear. There are many different ways of organization that can be applied in a hardware design process. Even though often forgotten during the planning phase, each project is only as good as the weakest team member.

- *Forms of organization*: Starting from small groups which are exclusively working for one project (task force), in larger teams some people might be involved in different projects that are headed by different project managers. Conflicts may results from the unclear responsibility.
- *Motivation of team*: The topic is not directly related to hardware design, but practice shows that many project managers - even though having very high technical skills - are not very competent when leading a group. Psychological and social components play a very important role. To keep the group members motivated the planning has to be realistic, competences must be clear, each individual most know about its importance and must participate in the success of the project. The project manager has to identify conflicts and has to solve them within the team without having "winners" and "losers" in the end.
- *Workflow management*: The classical software development models for the organization, like stepwise, sequential, or cyclic, can also be applied in the hardware domain. Techniques that have been developed along new methods for object orientation, like *tool-smithing*, can be applied when using HDLs having the same features.
- *Documentation*: Even though not very popular, the documentation of the individual steps in a project is very important. Looking at principles like design reuse the availability of a complete description becomes crucial. Often, components are still in use for years even after the designer left the company. Having a standard for documentation in this case helps significantly to guarantee a high quality of the modules.

The organization form determines the long term success of a hardware development group. Looking at the availability of hardware designers on the market, each company has to ensure good working conditions to keep the teams motivated. To guarantee a successful development it is important to introduce the different aspects of the project to all members of the team. For this, first a list of different project aspects is given (the list is not complete, but covers the most relevant topics) including some questions that should be answered **prior** to the project start:

- *Strategic goals*: In-house or outsourcing? Type of project? (consulting/coaching, turn-key development, joint venture)
- *Product / goals definition*: What exactly do we build?
- *Timeline / Duration*: When will the project start? How long will it take, and who will be involved in the different phases?
- *Resources*: Which and how many sub-tasks are defined? What is the number of people involved in each phase? How much do we get/invest for the project?
- *Complexity / difficulty*: How many tasks are defined and how are their mutual dependencies? Is the technology needed known or is research required? What is the risk in the calculations on investment?

[1] Of course, the checking of the modules can be done by the designer himself/herself, but he/she should not be involved in the controlling of the overall project.

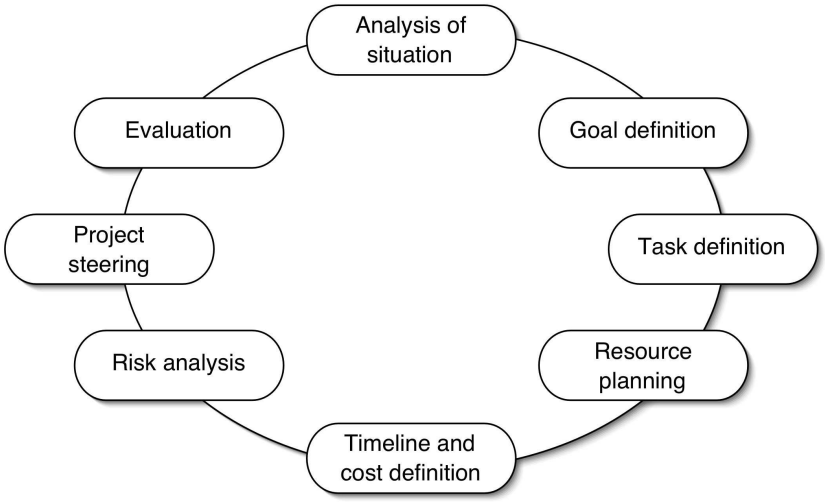

Figure 3: Project Management Tasks

- *Priority*: How does the success of the project influence other projects or sub-tasks? Has the project a strategic position?
- *Costs*: Can the costs and the risk of the project be estimated? Can further studies or information give a better picture?
- *Coordination/Organization*: Who is responsible for the defined sub-tasks and who manages the complete project?
- *Risk analysis*: Which potential items might endanger a successful project outcome? How can the tasks be prioritized and how to establish alternatives?

Some of the aspects mentioned seem to be obvious, but in practice it happens too often that some (or even most) of the questions cannot be clearly answered. It is also easy to see that not all points can be optimized at the same time. E.g. the costs become lower, when the project takes less time, but this usually has a negative influence on the quality of the product. Especially, the missing clear definition of the responsibility with respect to project coordination and organization in the early phase often results in serious problems later. A comprehensive understanding of the technical, organizational and social aspects of any project help to make it successful. Following the developments in software engineering, the cost-effectiveness of large-scale projects will become more and more important. In software engineering, using source lines of code (SLOC) was one of the earliest, straightforward methods to estimate the effort of a software project. As timeliness was one key argument against the use of SLOC, because this number can only be calculated after a project has been finished, there are further problems with SLOC. There is no relation to complexity of functions or numerical equations. A term that occupies three lines in the code, but took 4 weeks to be developed by experts counts as much as a three line code for an input/output device. As an organizational obstacle, the number that refers to the estimation is hard to understand for non-experts, i.e. top-level management. In more advanced cost estimation models, most of the attributes are also weighted according to their priorities. This leads to a complex estimation which outperforms SLOC methods. Function points (FP), first introduced in (Albrecht and Gaffney, 1983), has been developed historically for business applications. FPs are a weighted sum of the number of inputs, outputs, user inquiries, files, and interfaces to a system. How to transfer these cost estimation models to hardware projects is still an open research question.

3.4 Quality Management

As for software, it is a difficult task to measure the quality of a description in a HDL. The coding guidelines used in companies are not sufficient. Following the criteria well accepted in the software domain, some measures for hardware are outlined. We now distinguish between the goals that should be achieved and the methods and standards used:

Goals: It is obvious that the quality should be as high as possible, but it is also critical to consider the available resources. The following criteria have to be considered: correctness, reliability, user friendliness, reuseability, efficiency, and portability. Quality goals

have to be defined and methods have to be developed that ensure that these goals are met. It is not sufficient to just define a process and hope that the quality will be satisfying. Even though, some of the criteria above might be contradicting, a careful analysis what is expected and what can and should be created leads to a robust development flow.

Methods: To guarantee a high quality some measures or standards have to be defined. The currently most popular ones in the software domain are the Capability Maturity Model (CMM) and ISO9000. In his CMM, (Humphrey, 1995) distinguishes between five stages of organizational development: 1. Initial (ad hoc, reactive), 2. Repeatable (intuitive, depending on isolated staff, using former experience), 3. Defined (qualitative, defined process, controlling along the entire software process), 4. Managed (quantitative, development and use of metrics, data collection and analysis), 5. Optimizing (improving and adjusting the entire process). He also points out the steps that have to be made in order to improve the development process, which include understanding the current status of their development process and developing a vision of the desired process. This is the source of a potential conflict: the persons that measure are also the ones being measured. If the assessment reveals, that the stage the development processes are in is not the one expected to be but rather below that, the management is responsible. Accordingly, this implies the danger, that evaluations might be forged, so that the management appears in a better light. One way to avoid this conflict is to make use of outside consultant, whose perspective on that is neutral and not disturbed by such influences. A big disadvantage of such a procedure is, that is is costly and takes a lot of time, as outsiders have to become familiar with the present circumstances. Therefore, it might be helpful to ask a person within the company who is not directly involved in the development processes that are to be discussed. Again, this contains potential for a conflict, as the unit concerned might not see the necessity why somebody should get an insight is not really involved. It is human pride and maybe stubbornness that account for that. Finally, this remains a difficult and incalculable problem as it is often the case when human factors are involved.

Standards: Also in the hardware domain, using standards has been observed as being very important and resulted in HDLs, like VHDL or Verilog, for the synthesis step. In other domain, like "verification languages" standards are not established yet (see e.g. (Goering, 2001))[2]. But the certification of a development process is not used. Many of the rules developed for ISO900X on quality management and quality assurance can be directly transferred to hardware, while some aspects need to be modified.

To provide a high quality in the development process, clear goals have to be defined and methods have to be developed that ensure the realization of these goals.

4 PROBLEMS IN HARDWARE DESIGN

Beside the "optimal project flow" described so far, several problems can occur that should be considered when starting a hardware development project, although not unique for hardware projects.

- *Uniqueness of hardware systems*: Most hardware systems are only developed once and later on modified to meet new requirements. This often makes the requirement analysis and risk management (costs, financing etc.) difficult.
- *Technically oriented management*: Many (maybe most) of the project managers have a technical background and have been designers before. They are very qualified regarding the hardware itself, but do not necessarily have good management skills in leading a group. Often they start to design on their own or try to influence the designers in their group. This might leads to problems regarding the role of each team member. (On the other hand "pure managers" without sufficient insight in the technology have problems of getting accepted by the design teams!)
- *Weak planning*: In practice the time invested in the early project phase for specifying the details of the behavior of the hardware to build is not sufficient. An unclear specification is one of the main reasons for delayed or even failing projects.
- *High number of possible solutions*: Usually, there is not one unique way of realizing a circuit. The main bounds are given by the constraints of the resources of the hardware and are very difficult to restrict regarding nowadays hardware complexity.
- *Individuality of designers*: Many designers see themselves more as artists than as hardware designers. They are individualists and are often not willing to work in a team. This is especially something they do not learn during their education at the universities. Even though this is changing

[2]Even though recently SUGAR has been selected as a property language standard there are still more than 10 other languages around.

more and more, as group oriented working becomes a topic.

- *Rapid technological changes*: The technology evolves very fast, and often the tools and techniques have to be modified during the development cycle. This makes an exact planning regarding the resources nearly impossible. The project managers must have flexibility to incorporate these and must also have the leader quality to communicate these "negative messages" to the project team.
- *Missing standards*: Even though there are many standardization committees in the hardware domain, there are still many problems related to incompatibility of data formats or language definitions.
- *Status monitoring*: Many systems cannot be tested under "real" conditions until the final design is done. Thus, during the project it is often difficult to say what the current status of the project is. A project might seem in time, since most of the code is written, while a lot of effort is needed for debugging, if the quality is poor.

5 CONCLUSIONS

Over the last 25 years the number of components of an integrated circuit has grown from a few thousand to more than 20 million in nowadays ASICs. While in the beginning of this process one or two designers were doing the complete development, today in a project group more than 30 designers are working in parallel. This demands for a closer integration of project engineering and hardware design. Problems that might occur during a chip design have been discussed in this paper. Hardware designers can make use of many concepts that have been developed over the last couple of years in the software domain. This transfer is justified by the similarities between hardware and software development. Only by applying structured methods of project management high quality designs can be obtained and successful tape-outs of multi-million gate ASICs can be expected.

REFERENCES

A. J. Albrecht, J.E. Gaffney, Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation, IEEE Transactions on Software Engineering, (9)6, pp.639-648, 1983

B. Bentley, Validating the Intel Pentium 4 Microprocessor, Design Automation Conference, pp. 244-248, 2001

B. W. Boehm, A spiral model of software development and enhancement, IEEE Computer, 21(5), pp. 61-72, 1988

Y. Chu, An Algol-like Computer Design Language, Comm. ACM, p. 607-615, October 1965.

G. DeMicheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994

T. DeMarco, Controlling Software Projects: Management, Measurement and Estimation, Prentice-Hall, 1982

C. Floyd, A Systematic Look at Prototyping, in R. Budde, K. Kuhlenkamp, L. Mathiassen, H. Züllinghoven (editors), Approaches to Prototyping, pp. 1-9, Springer, 1984

R. Goering, Assertion's Babel Tower? EE Times, August, 2001, available online under url: http:www.eedesign.com/story/OEG20010828S0054

W. Humphrey, A Discipline for Software Engineering. Reading, Addison-Wesley, 1995.

G. Hachtel, F. Somenzi, Logic Synthesis and Verification Algorithms, Kluwer Academic Publisher, 1996

IEEE Standard 1219: Standard for Software Maintenance. Los Alamitos CA., USA. IEEE Computer Society Press, 1993.

N. Krishnamurthy, M. Abadir, A. Martin, J. Abraham, Design and Development Paradigm for Industrial Formal Verification CAD Tools, IEEE Design & Test of Computers, pp. 26-35, July-August 2001

M. Keating, P. Bricaud, Reuse Methodology Manual, Kluwer Academic Publisher, 1999

C.F. Kemerer, Software Project Management, Readings and Cases, McGraw-Hill, 1997

B.P. Lientz, E. Swanson, Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations, Addison-Wesley, 1980

P. Naur, B. Randell (editors), Software Engineering: A Report on a Conference sponsored by the NATO Science Committee, NATO, 1969

Sommerville, I., Software Engineering (7th Edition). Pearson Addison Wesley, 2004

R. Vonk, Prototyping - The effective use of CASE Technology, Prentice Hall, 1990