CHAPTER 5

Towards Improvement of software Products for Parallel and Distributed Systems

Dr. Ahmed Taha Shehab El-Dean

&

Dr. Mohammed M. Eassa

ABSTRACT

Information technology is advancing at enormous rate, but most of software products are facing some problems. Due to the compexity of parallel/distributed systems; the development of their software products is affected by many factors and so it is eroor-prone and costly.

In order to achieve cost-effective with high quality; parallel/distributed software products; much more efforts must be done in all the phases of the development process specially the very beginning ones; the specification and design phases.

Software engineering and quality assurance techniques must be the foundation of the production of parallel/distrbuted software systems.

This paper introduces a framework that combines the software engineering and quality assurance techniques to improve software products for parallel/distrbuted systems. The proposed framework for the development of software products is applied on the data flow machine as a case study.

1 - INTRODUCTION

Computer hardware is improved from vacuum tubes to VLSI, but there has been no significant change in the sequential abstract computing model. The demand for ultrahigh speed computing machines is incrasing every day. The major difficulty in satisfying this demand in uniprocessing is the physical constrains of hardware and the sequential and centralized control in the Von-Neumann computing model. The

production of faster, more reliable, and cheaper electronic components; made it evident that the performance of uniprocessor computers is limited. The alternative to sequential processing is parallel procèssing machines. The transition from sequential to parallel and distributed computing offers high computing power that is very important to solve computationlly intensive problems [3,21].

Data flow machine is one of the most promising parallel processing machines. Conisderable progress is made in building data flow machines during the last few years. The data flow computing concept is the most effective, promising computing method to implement in machine architecture for high speed computing[21].

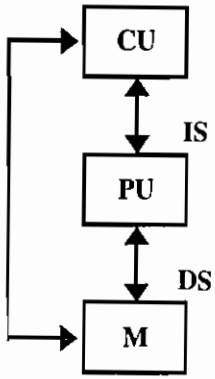
1.1 Motivation :

The production process of software products differs from the production of goods. The manufacturing costs of software products are neglected; it is the cost of copying a piece of software. Also, the maintenance process differs form software products to hardware products. In software products the maintenance process consists of two activities: correction and update. The corrective activity leaves the functional specification intact while the update acivity changes the functional specification of the software product. Really the maintenance process affects the early phases of the development process.

As the architecture of hardware becomes more and more complex; the supporting software products become more and more complex. As software products become more and more complex; the development process becomes more and more complex. As the development process of software product becomes more and more complex; software engineering and quality assurance tools & techniques become more and more vital to achieve cost-effective high quality product.

2. PARALLEL AND DISTRIBUTED PROCESSING PARADIGMS :

The majority of today's computers are conceptually very similar. Their architectures and modes of operation follow the basic design principles fromulated by John Von-Neumann and coworker [3]. The Von-Neumann approach is based on the simple idea of a control unit that fetches an instruction and its operands from a memory unit and sends them to a processing unit in which the instruction is executed and the result is sent to the memory. The Von-Neumann paradigm is called Single Instruction Stream Single Data steam (SISD) as shown fig. 2.1.



CU : Control Unit.

PU : Processing Unit.

M : Memory Unit.

IS : Instruction Stream.

DS : Data Streem.

Fig 2.1 : The Von-Neumann Computing Paradigm [3].

Computers operate by executing instructions on data. The computer architecture is classified on the basis of how the machine relates the instruction stream to the data stream into 4 classes:

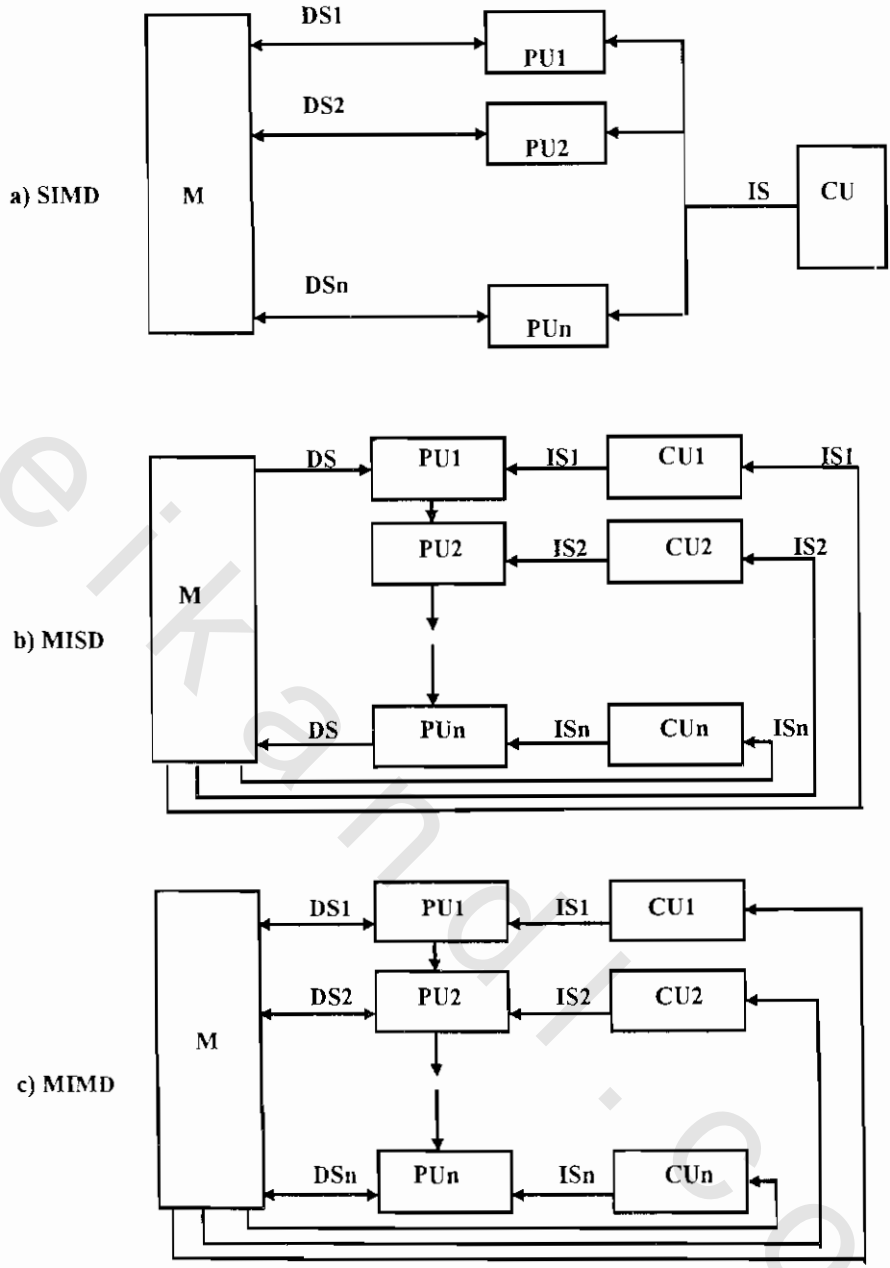
1 - Single Instruction stream, Single Data stream (SISD) Von- Neumann machines),

2 - Single Instruction stream, Multiple Data stream (SIMD),

3 - Multiple Instruction stream, Single Data stream (MISD),

4 - Multiple Instruction stream, Multiple Data stream (MIMD).

The architectures of SIMD, MISD, and MIMD are shown in fig. 2.2 (a), (b), & (c).



M: Memory, PU. Processing Unit, CU: Control Unit,

DS. Data Stream, & IS Instruction Stream

Fig. 2.2 Parallel processing architectures [3].

2.1 Data Flow Versus Contral Flow :

Data flow computers are based on the concept of data-driven computation that is different from the operation of the conventional Von-Neumann machine.

The main difference is that the instruction execution in Von-Numann is under program-flow control, while in a data flow computer is driven by data availabilty.

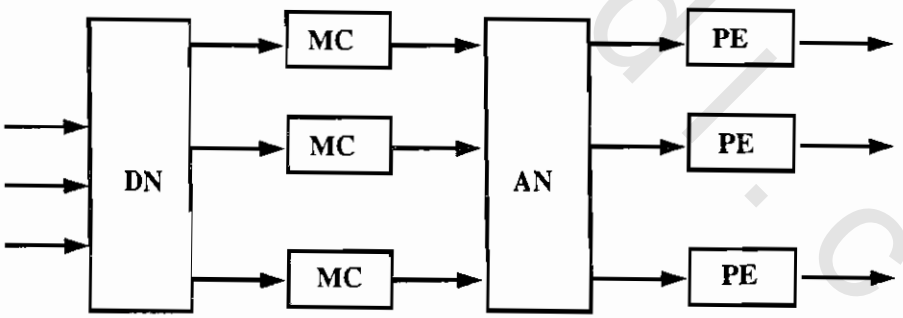
Data flow is a technique for specifying computation in a two-dimensional graphical form: instructions that are available for concurrent execution are written alongside one another, and instructions that must be executed in sequence are written one under the other. Information items in data flow computers appear as operation packets and data tokens. An operation is composed of the opcode, operands, and destination of its sucessor instructions. A data token is formed with a result value and its destination.

2.2 Data Flow Machine : [10, 21]

Data folw machine is one of the most promising parallel processing machines. Research and development on data flow computing have been started at 1968. There are two main classifications for data flow machines: Static and dynamic data flow machines [21].

MIT Static Data Flow Machine

One of the static data flow machines is MIT machine. The block diagram of the MIT static data flow machine architecture is shown in fig 2.3.



DN : Distribution Network, MC: Memory Cell, AN: Arbitration Network, & PE: Processing Element

Fig. 2.3 The architecture of the MIT data flow machine [21].

MIT Static data flow machine consists of four units:

- ***Memory unit*** : it is a collection of blocks; cell blocks; of central storage locations. Each block stores a node information which are operation, operands, and the destination addresses. The fist step to execute a program graph is to load it into the cell blocks. A part of the memory is used to store arrays to support rapid access of array elements.
- ***Processing section*** : it comprises eight identical processors. The instruction set contains floating point, fixed point, logical packet communication, and shift instructions.
- ***Arbitration network*** : it communicates the enabled packet of instructions from the cell blocks of the memory to the processing elements using a **FIFO** buffer.
- ***Distribution network*** : it communicates the results of execution from the processing elements to the cell blocks of the memory using a **FIFO** buffer.

MIT static data flow machine has one communication path of 32 bits width and 11 words instruction packets. If a ***processor is faulty***, the firing node is lost, i.e. no result will be conducted to the next node or other processors, so the computation is restarted from the beginning [21].

The execution of a data flow language program has the following steps:

- ***Data Flow Graph Construction***
- ***Parallelism extraction***: instructions that are available for concurrent execution are extracted.
- ***Program execution:*** the execution of a program is the firing of all of its data flow graph nodes. For each node to be executed the following steps are required:
  - *Node Reading* : The node information is read from the memory.
  - *Node Enabling* : if all of the node input tokens are available, the node is enabled to be fired.
  - *Node Firing* : the node is aassigned to an available processor to execute the operation associated with that node using the functional units of the processor; then the resultant token is passed into the output arc.
  - *Result Distribution* : the result is moved to the next successor nodes.

3. Software Engineering and Quality Assurance

3.1 Software engineering :

Software engineering is a discipline concerned with the production and maintenace of software products. As other engineering fields, software engineering concerns of the development of software products in an effective, economical, and timely fashion. Software engineering is based on the foundation of computer science, management science, and engineering approach to problem solving.

The main goal of software engineering is to achieve the right balance among quality, productivity, and cost effectiveness of software product [6, 7, 8, 16, 17, 21].

3.2 Software Quality :

There are many definitions for the quality of products generally, and the quality of software products specifically. The following are some quality defintions [19]:

- **From the manufacturing point of wiew;** the quality means conformance to requirements
- **From the usability point of wiew;** the quality means the intensity of failure (reliability).
- **From the product value point of wiew;** the quality means the degree of excellence at an acceptable price.
- **From the customer point of wiew;** the quality is the capacity to satisfy wants.
- **American National Stanard Institute;** the quality is the totality of features and characteristics of a product or service that bears on its ability to satisfy given needs.

The definition of software quality can be applied in the development process of the software product as in defintion 1; as well as; in the operation phase as in the definitions 2, & 4.

The most serious problems facing the software industry are the low-quality problems.

The Software Engineering Applied Technology Center (**SEATC**) for the National Security Agency (NSA) addresses software quality as a key initiative to imporove software development [26].

3.3 Software Quality Assurance (SQA) :

Software quality assurance is the methodologies and tools of preventing or/and

correcting the deviations from the requirements and measuring the software quality factors [2, 19, 26].

Testing is the primary tool of software quality assurance. Quality assurance concerns not only performing a test, but also by defining the software products quality factors, establishing standards for test, and introducing methodologies for test design and test cases generation [7, 16, 17, 26].

Software quality assurance must be conducted; in a specific methodology, to each phase of the software development process as well as to the final software product to achieve the total quality of the software products [19, 26].

Software quality assurance holds great promise for improving the practice and moving the profession along the path toward a full software production engineering discipline [16, 17].

A proposed general model for quatlity assurance is shown in fig. 3.1

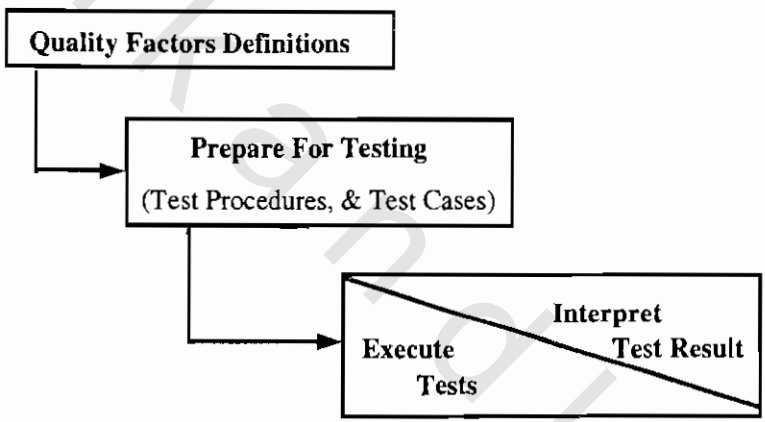


Fig. 3.1 : The proposed quality assurance model.

4. THE INCREMENTAL DEVELOPMENT OF SOFTWARE PRODUCTS

The development process of software products is an engineering activity; so it is classified into a set of phases [22]. The waterfall model; shown in fig 4.1; presents the software products life cycle.

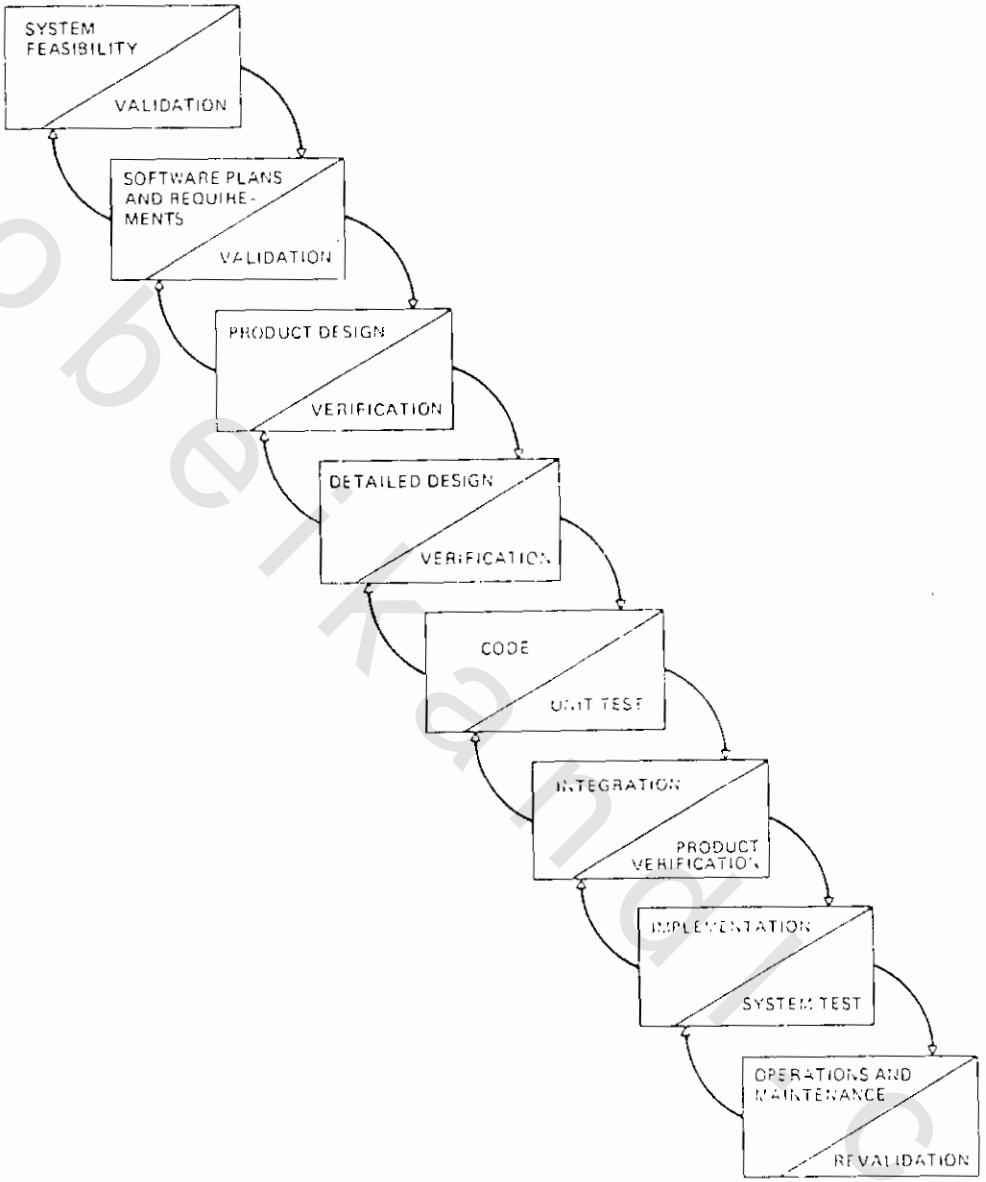


Fig. 4.1 : The "Waterfall" model of the software life cysle [4].

In fact it is cost-effective to modify the sequence of software life cycle. Incremental development of software product is one of the most important Waterfall refinements.

B.W. Boehm introduced a refinement to the Waterfall model of software development process; that is the incremental development of software products [4]. The

incremental development means that the software is developed in increments of its functional capability. The advantages of incremental development are the testability of increments and the cost effectiveness of product refinement. The Waterfall model using the incremental development is shown in fig. 4.2.

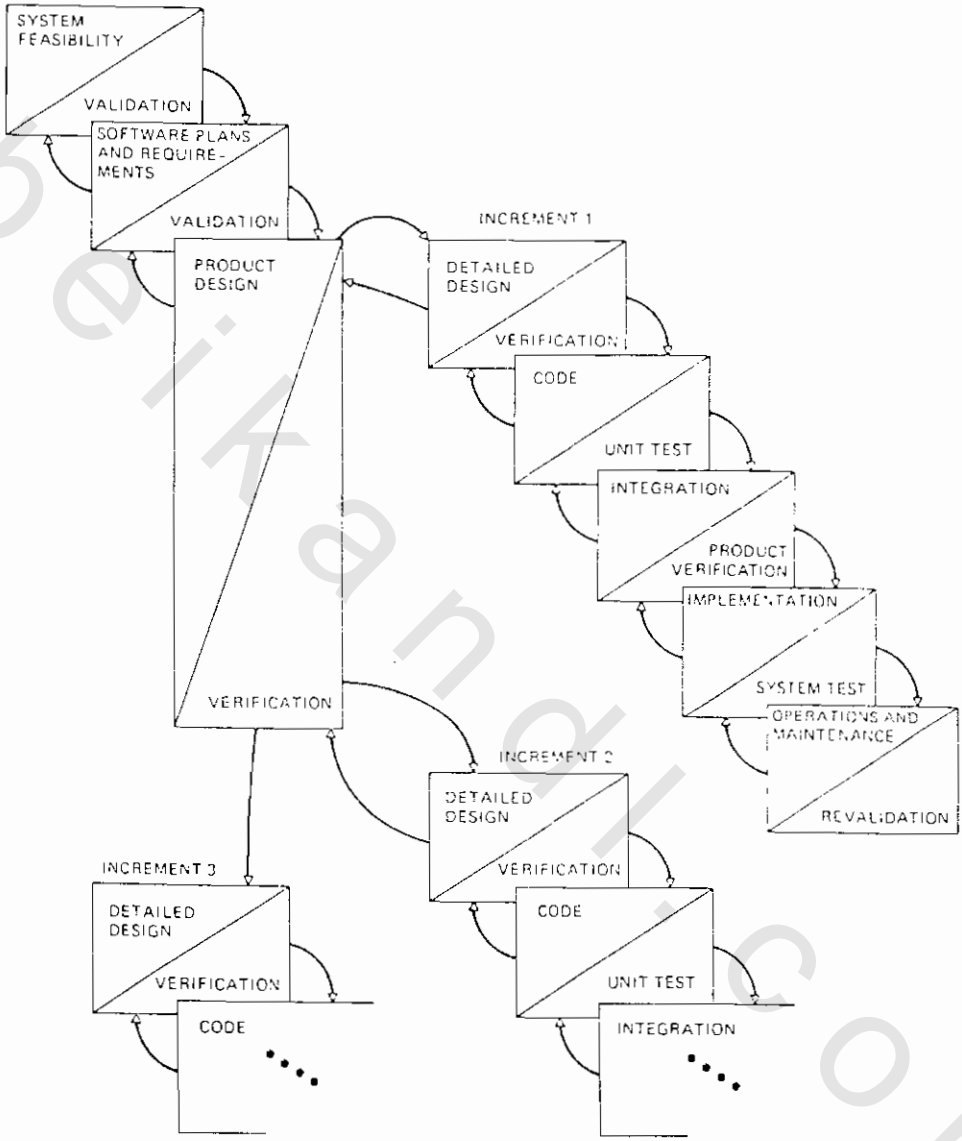


Fig. 4.2 : The Waterfall model using the incremental development [4].

♦ **The Incremental Formal Specification of Software Products :**

The incremental development as shown in fig. 4.2 considers the detailed design phase of the development as the base to start the increments of the development process.

In fact the specification is the basis of development, also it provides the communication between the user and the developer and it often serves as a standard document across the remainder phases of development. The specification phase of the development process is very important since the specification errors, if not fixed very early, need a great effort and cost to fix [6, 7, 11, 15, 16, 17, 18].

Today software specification is considered as an independent area of study because of its importance to the engineering discipline. Specification leads to proposals for simplification, which in turn lead to a system that is easier to design, implement, and maintain [15].

Formal methods for specification provide the bases of specification quality assurance, and so increasing the confidence in the developed software [15, 22]. Unfortunately, most of software industries use informal methods for specification. Informal specifications may be easy to read and understand, but they are often incomplete, vague, ambiguous, and can not be quality assured. Formal specification must take into consideration the structure and the behavior of the specified system [1].

Formal behavioral specifications consider three perspectives:

♦ cause-effect specification,

♦ time specification, and

♦ temporal specification.

Each of the three perspectives has its tools and techniques to specify systems and to define the quality factors for that specification.

Comlpete behavioral specification must posses the three behavioral specification perspectives. The incremental specification process must consider the three perspectives as formal increments of the behavioral specifcation as well as the structural specification and the informal behavioral specifications. The informal specification is needed as explanations for the formal specification. The complete incremental model of software development is shown in fig. 4.3.

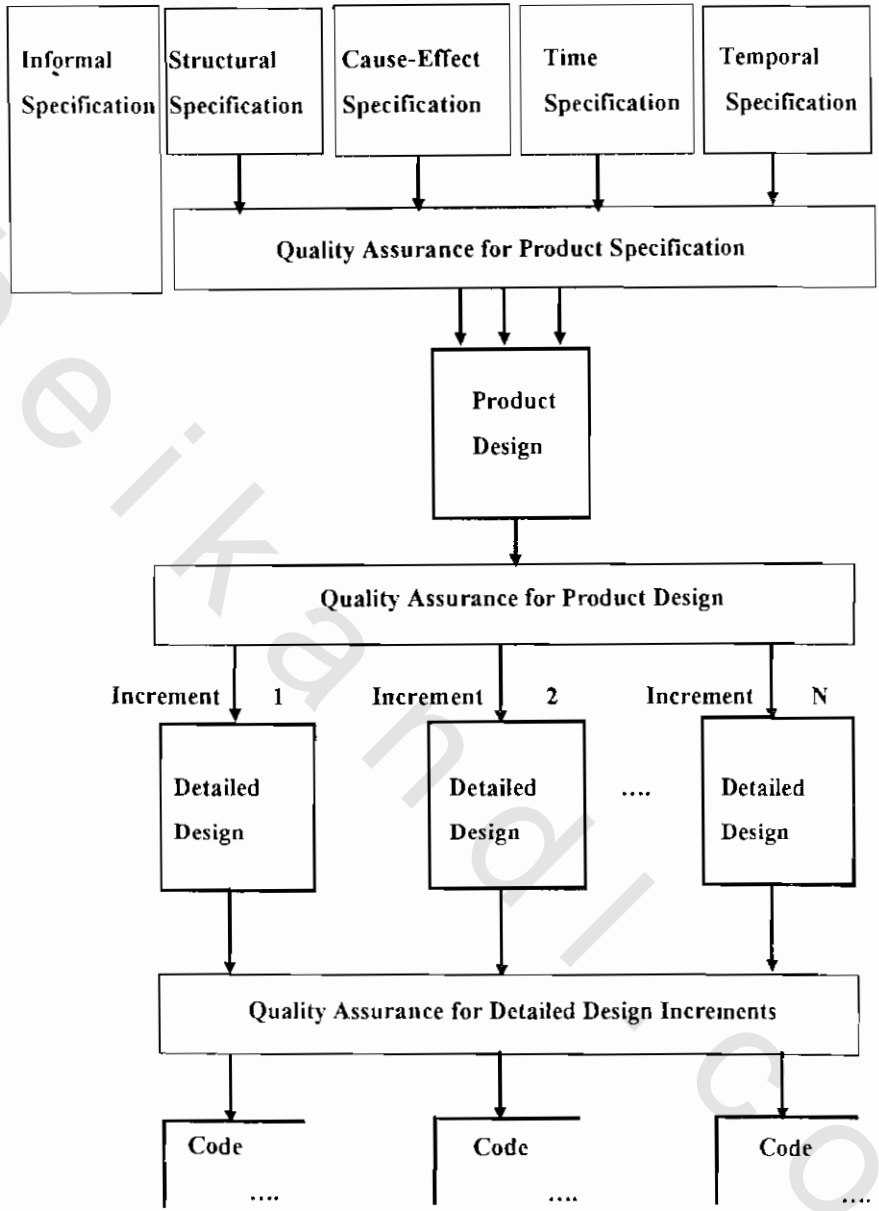


Fig. 4.3 : The complete incremental model of software development.

5. INCREMENTAL DEVELOPMENT FOR DATA FLOW MACHINE SOFTWARE

The development process of data flow software products can be improved by adhering the incremental model of software development. The main problem is the extraction of parallelism [10].

There are three levels of parallelisms :

- Program level : independent programs are executed in parallel,
- Task level : programmer specified procedures (tasks) are executed in parallel, and
- Instruction level : independent operations within an expression are executed in parallel, independent serial sequences are executed in parallel, and independent functions are executed in parallel.

Accoding to the complexity of the structure and the behavior of the data flow systems; there is a need for the incremental specification to specify all the levels of parallelism accurately and completely.

Applying the Incremental Specification [1. 12, 20, 21]

- **Cause-effect specification** : the interaction among the components of the system (at each level of parallelism) is specified using **Petri Net** as a formal specification tool. The performance quality assurance factors are the safeness and liveness. The reachability analysis is used as a testing methodology to asses the quality assurance factors.
- **Time specification** : the amount of time that can be taken by each of the interacting components is specified using the **PERT chart** as a formal specification tool. The performance quality assurance factor is the minmum required time for the interacting components to terminate. The Critical Path Method (**CPM**) is used as a testing methodology to asses the time requirements. Also Timed Petri Nets (**TPN**) can be used to specify the real time interaction among concurrent processes.
- **Temporal specification** : The temporal relations among the interacting processes are specified using Temporal Logic (**TL**). The performance quality assurance factors are the availability, safeness, and liveness. The theorem proving methodology is used to asses the quality assurance factors.

6. CONCLUSION AND FUTURE WORK

The main goals of this paper are :

- to introduce the concept of formal incremental specification,
- to present a quality assurance model for software development process, and
- to improve the increment development of software products by refining the specification and conducting the quality assurance model to all the development phases.

In this paper software engineering, and quality assurance tools and techniques are considered the foundation of the development of software products improvement.

The formal incremental specifications means that the specifcation must be classified into increments : cause-effect specification increment, real time specification increment, temporal specification increment, and structural specification increment.

The incremental specification is considered a comprehensive specification of the software product as the base of the development process. Applying the quality assurance model to all the software development phases represent the foundation to the software product total quality.

As one of the parallel processing machines; data flow machine is considered as a case study to apply the formal incremental specification and the quality assurance model. The specification and quality assurance tools and methodologies for the data flow machine are presented.

Much more efforts must be done to define new quality assurance factors and their testing methodologies for the tools of each of the specification perspectives.

Much more effort must be done to produce automated tools for the formal incremental specification and quality assurance specifically and the development life cycle generally.

REFERENCES AND BIBLIOGRAPHY

[1] Ahmad Taha Shehab El-Dean, ***"Hardware Testing Temproral Logic"***, Ph. D. Dissertation, Computer & Systems Eng., Al-Azhar Univ., March 1997.

[2] Alan Wood, ***"Predicting Software Reliability"***, IEEE COMPUTER SOCIETY, NOVEMBER 1996.

[3] Albert Y. Zomaya, Editor, ***"Parallel and Distributed Computing Handbook"***, McGraw-Hill 1996.

[4] B. W. Boehm, ***"Software Life Cycle Factors"***, HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc, 1984.

[5] Bernard P. Zeiglar, ***"Theory and Application of Modeling and Simulation : A Software Engineering Perspective"***, HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc, 1984.

[6] Boris Beizer, ***"Software Performance"***. HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc, 1984.

[7] Boris Beizer, ***"Software Systems Testing and Quality Assurance"***, Van Nostrand Reinhold Company Inc, 1984.

[8] C. R. Vick, ***"Introduction : A Software Engineering Environment"***, HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc, 1984.

[9] Daniel Jackson, and Jeannette, ***"Lightweight Formal Methods"*** IEEE COMPUTER SOCIETY, APRIL 1996.

[10] Don Oxley, Bill Suber, & Merrill Cornish, ***"Software Development for Data Flow Machines"***. HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc, 1984.

[11] Edward F. Miller, ***"Software Testing Tehnology : An overview"***, HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc, 1984.

[12] Fathey E. Essa, ***"The use of Temporal Logic in Software Testing"***, Ph. D. Dissertation, Computer & Systems Eng., Al-Azhar Univ., 1988.

[13] Gregory A. Hansen, ***"Simulating Software Development Process"***, IEEE COMPUTER SOCIETY, JANUARY 1996.

[14] Hon H. So, ***"Graph Theoretic and Analysis in Software Engineering"***, HANDBOOK of SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc., 1984.

[15] Castro, ***"Distrbution System Specification using A Temporal-Causal Framework"***, Ph. D. Dissertation, London Univ., October 1990.

[16] John D. Musa, ***"Software Reliability-Engineered Testing"***, IEEE COMPUTER SOCIETY, NOVEMBER 1996.

[17] John D. Musa, ***"Software Reliability"*** HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc., 1984.

[18] K. H. Kim, ***"Software Fault Tolerance"***, HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc., 1984.

[19] M. L. Gabri, ***"Total Quality Management"***, Hanns Seidel Foundation, Management Traning Project, Cairo, 1993.

[20] M. M. El-Boraey, ***"Reliability of Parallel Processing Systems"***, Ph. D. Dissertation, Computer & Systems Eng., Al-Azhar Univ., 1992.

[21] M. P. Mariani, & D. F. Palmer, ***"Software Development for Distrbuted Computing Systems"***, HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc, 1984.

[22] Michael J. Lutz, "Consumable Mathematics for Software Engineering", IEEE COMPUTER SOCIETY, APRIL 1996.

[23] R. D. Williams, ***"Management of Software Development"***, HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc., 1984.

[24] R. W. Wolverton, ***"Software Costing"***, HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc., 1984.

[25] S. H. Saib, ***"Formal Verification"***, HANDBOOK OF SOFTWARE ENGINEERING, Van Nostrand Reinhold Company Inc., 1984.

[26] Thomas Drake, ***"Measuring Software Quality : A Case Study"***, IEEE COMPUTER SOCIETY, NOVEMBER 1996.