International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:12, 2009

A New Divide & Conquer Software Process Model

First A. Hina Gull, Second B. Farooque Azam Third C. Wasi Haider Butt, Fourth D. Sardar Zafar Iqbal

***Abstract*—**The software system goes through a number of stages during its life and a software process model gives a standard format for planning, organizing and running a project. The article presents a new software development process model named as "Divide and Conquer Process Model", based on the idea first it divides the things to make them simple and then gathered them to get the whole work done. The article begins with the backgrounds of different software process models and problems in these models. This is followed by a new divide and conquer process model, explanation of its different stages and at the end edge over other models is shown.

***Keywords*—**Process Model, Waterfall, divide and conquer, Requirements.

I. INTRODUCTION

SOFTWARE Development Lifecycle is the structure imposed on the development of Software product"[1]. Software system goes through a number of stages during its life, from requirements elicitation to deployment and maintenance. A software process model gives a standard format for planning, organizing and running a project. There are number of models for software processes to work in order, describing different flows and approaches for activities which are the part of that process. A process model is chosen by keeping in view the nature of the project, tools to be used, and deliverable that is required [2].

II. TRADITIONAL PROCESS MODELS:

Below is the brief introduction of several process models:

A. Build and Fix Model:

The software process models history begins with the introduction of a model called "Build and Fix". The model has only two steps:

- Write the Code
- Fix problems in the code.

Thus, the main theme of the model was to write some code first and then think about different phases of development [3].

The model suffers from the following flaws:

- Due to a number of fixes, the resulting code had a poor structure and these fixes were highly expensive. This was due to the absence of a detailed design phase before the coding phase [8].
- Does not follow any proven method. Its working included, first coding then moving towards other stages like Requirements, Design, Test & Maintenance. Due to which the resulting product had a structure which often did not meet the requirements of the user consequently ending up in either project termination or redevelopment which was highly expensive. [10]
- Not suitable for environment where changes are dynamic in nature [4] .
- There was no specific stage for testing. Coding phase included a small module for testing. Due to which code was poorly tested.[3]

Above mentioned reasons stressed upon the introduction of phases like Requirement, Designing, Coding, Testing, Maintenance etc.

B. Waterfall Process Model:

First, a stepwise sequential model was presented in 1956, but that was not a formal model for development. The first ever formal description of Waterfall model was given in 1970 by Winston W. Royce. The model formed the basis for most software development standards and consists of the following phases: Requirements specification, Design, Construction, Integration, Testing and Debugging, Installation and Maintenance.

To follow this model, developer has to move in sequential manner i.e. one has to complete a phase fully and then have to move in sequential fashion [4]. The main disadvantages in this model were:

- Rigid design and inflexible procedure [4].
- Restricts Development of software by blocking movement back to a prior stage, that is, it restricts looping back to prior stages even if new changes surface which need to be accommodated. [8]
- The requirement stage constituted gathering concrete specifications including both vague and some critical requirements collected together. As the requirements were frozen before moving to the design phase, using the incomplete set of requirement, a complete design was worked on. So, was the case with the code phase. Such an approach worked normally well for a small project requiring average amendments. In case of a large project, completing a phase and then moving back to reconstruct the same phase, incurred a large

F. A. Author is with the College of Electrical and Mechanical Engineering, National University of Science and Technology Rawalpindi Pakistan (phone: 092-343-5203608; e-mail: hina.gull03@gmail.com)
S. B. Author, is with the College of Electrical and Mechanical Engineering, National University of Science and Technology Rawalpindi Pakistan (phone: 092-51-9278056; e-mail: farooq@ceme.nust.edu.pk)
T. C. Author is with the College of Electrical and Mechanical Engineering, National University of Science and Technology Rawalpindi, Pakistan (phone: 092-333-5782107 email: butt.wasi@gmail.com)
F. D. Author is with the Department of Computer Sciences, Iqra University of Science and Technology Islamabad Pakistan (phone: 092-343-5267319; e-mail: zafar.iqbal38@gmail.com)

overhead. Use of frozen requirements was a worst approach as compared to the prototype approach giving the user immediate "look and feel" of the system in development. It could not be used for the interactive end user applications. Analysis & Design are the kind of Phases whose completion from different dimensions is not possible; hence we can never really say when we are done with these phases. [4]

- It emphasizes on the use of fully elaborated documents for completion criteria of requirements and design phases. It is unnecessary to write elaborated specification for a product before implementation.[3]
- Waterfall Model faced *"inflexible point solutions"* which meant that even small amendments in the design were difficult to incorporate late in design phase.
- Once a phase is done, it is not repeated again that is movement is from one phase to the next and the opposite is not supported. Deadlines are difficult to meet in case of large projects. [10]

C. Prototype Model:

In this model, the prototype for the system to be developed in built, tested and reworked as necessary. With this approach the development begins with the most visible aspect of the software system, for which the prototype is developed. Then the development continues when the feedback from this prototype is received. The process is found to be useful for the systems where the requirements are changing rapidly. The process begins with the requirements gathering phase, a quick design then occurs which then leads to the development of prototype. The prototype is then evaluated by users and customers and is reworked until the customer and users are satisfied. The prototype can be problematic at the following points:

- The main disadvantage is that it is not known at the start of the project that how long it will take to create a product which is acceptable to the users. Also how much iterations it will take to make an acceptable product [6].
- The premature prototypes lack key consideration like security, fault tolerance, distributed processing and other such key issues. Such requirements jeopardize the project as these initial incomplete prototypes having weak architecture cannot be enhanced to achieve the key consideration [10].
- Developers are in such a rush that they hardly consider all the functionalities of the prototype. In order to release the product as soon as possible, the prototype with some additions is released on or before the target release date. This happens due to lack of user analysis activities; the end product contains features the user is hardly aware how to use. [3]
- Often the developers make implementation compromises in order to make the prototype work quickly, which will lead to the use of inappropriate operating system or programming language [7].

D. Incremental Development Model:

The model develops the system in small increments. In first step all the requirements are gathered and then the subsets of the requirements are assigned to each increment or release. The increments are developed in sequence leading to the end product at the end of the last increment. It reduces the overall effort and also provides system earlier to the user. The main feature of this model is that we have less effort on coding while more emphasis is on requirements gathering and analysis [8]. The disadvantages of the model are:

- It is difficult to map requirements directly to different increments. Include excessive user involvement. Poorly defined scope as scope of the product may vary increment to increment.[6]
- After every iteration, the user gets a "look & feel "of the system. An overhead in the model is rapid context switching between various activities. Evaluation after each iteration involving user involvement consumes a lot of time. [9]
- Identify key issues starting from the early iterations, not waiting for later iterations. Focus appropriately starting from the first iteration to the upcoming ones. Use results of the early iterations to manage the risks of the project. [9]

E. Spiral Model:

Spiral model is an evolutionary model that combines some aspects of prototype model and some aspects of linear sequential model. The model is divided into some task regions, which are as follows: Customer communication, Planning, Risk Analysis, and Engineering, Construction and release and Customer evaluation.

Major distinction of this model with others is having the risk criteria at every stage. This model is also known as risk driven model because it identifies the risk areas and sources. The model is divided into cycles and each cycle ends up with the end of an activity in which the risk analysis is the major factor. The model is used mainly for large projects. It uses a organized stepwise procedure, like the classic life cycle model, but adds it into an iterative development framework that more mirrors the real world [11].

The following disadvantages are identified in this model:

- The Spiral Model performance depends on the risk assessment expertise of the involved software team. If risk analysis is poor the end product will surely suffer.
- Great care is taken by software developers to identify and manage resources of the project identifying aptly all possible risks making the spiral model people dependent. Another difficulty of the spiral model is adjustment of contract deadlines using the spiral model.
- A number of risks, constraints, alternatives, models etc. need to be analyzed but never are these risks or objectives listed and no specific risk analysis technique is mentioned. Software developers begin with the vague idea of risk analysis according to their expertise.
- For large projects expert software developers can

produce efficient software products but in case of a complex large project absence of specific risk analysis techniques and presence of varying expertise can create a chaos. The model also demands considerable risk expertise. [11]

F. Win Win Spiral Model:

The original spiral had four sectors, beginning with "Establish next-level objectives, constraints, alternatives." The two additional sectors in each spiral cycle, "Identify Next-Level Stakeholders" and "Identify Stakeholders' Win Conditions," and the "Reconcile Win Conditions" portion of the third sector, provide the collaborative foundation for the model [11].

- The main drawback of the model is that each successive cycle begins after a detailed risk analysis imposing many different constraints, objectives & alternatives. But never are these risks specifically mentioned and vary project to project [3].

G. Rapid Application Development Model:

The model is considered to be incremental development model and that have emphasis on short development cycle. This model is called rapid application development because in it rapid application development is achieved by using component based development. The model has the following phases: Business Modeling, Data Modeling, Process Modeling, Application Generation and Testing and Turnover. Like all other process models, the Rapid Application Development has the drawbacks:

- Reduction in scalability is because an application developed by following RAD begins as a prototype and evolves into a finished application.
- Reduction in features occurs due to time boxing, where features are given to later versions to finish a release in a small amount of time.
- For large projects, RAD requires a sufficient number of human resources to create a right team. Also RAD is not suitable for all types of application development. If the system cannot be modularized properly building the components for RAD will be problematic [12].

H. Rational Unified Process:

"RUP provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget. The Rational Unified Process activities create and maintain models. Rather than focusing on the production of large amount of paper documents, the Unified Process emphasizes the development and maintenance of models—semantically rich representations of the software system under development. The Rational Unified Process provides each team member with the guidelines, templates and tool mentors necessary for the entire team to take full advantage of among others the following best practices: Develop software iteratively, Manage requirements, use component-based architectures, visually model software, Verify software quality, Control changes to software. The process can be described in two dimensions, or along two axis:

- The horizontal axis represents time and shows the dynamic aspect of the process as it is enacted, and it is expressed in terms of cycles, phases, iterations, and milestones.
- The vertical axis represents the static aspect of the process: how it is described in terms of activities, artifacts, workers and workflows.

The software lifecycle is broken into cycles, each cycle working on a new generation of the product. The Rational Unified Process divides one development cycle in four consecutive phases"[13].

- Inception phase
- Elaboration phase
- Construction phase
- Transition phase

Each phase is concluded with a well-defined milestone—a point in time at which certain critical decisions must be made and therefore key goals must have been achieved. Each phase has a specific purpose". The identified drawbacks of the process are:

- Each phase has a milestone which needs to be satisfied for the next particular phase to start.
- If the respective milestone of the particular phase is not satisfied the entire project might get cancelled or re-engineered before proceeding further.
- The satisfaction criteria of a particular milestone has its own constraints and are not listed specifically [13].

İ. The V-Model:

It is assume to be the extension of Waterfall Model. The difference is that it doesn't move in linear way, instead the process steps are bent upwards after the coding phase to form V-shape. Each phase has an associated tasting phase. It consists of following phase:

Verification phases:

1. Requirement analysis
2. Architecture design
3. Module design

Validation phase:

1. Unit testing
2. Integration testing
3. System testing
4. User acceptance testing

It has the following drawbacks;

- It addresses software development within a project rather than a whole organization.
- The V-Model is not complete as it argues to be, as it covers all activities at too abstracts level.

III. Divide & Conquer Model

The main divide and conquer model is as follows:

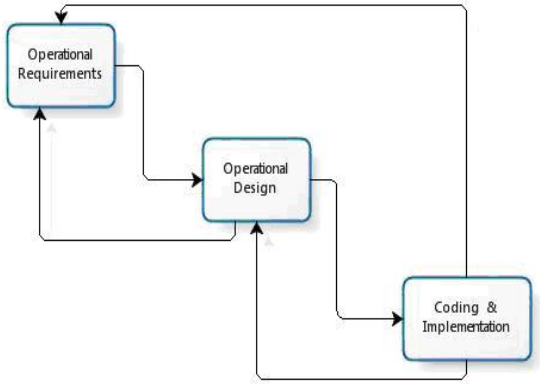

Fig. 1 Divide & Conquer Model

The model has mainly three phases, Operational Requirements, Operational Design and Coding and Implementation. Each phase is further elaborated and parts of each phase are shown in more details in the coming sections. The model suggests that the movement starts with operational requirements and then can go to operational design. But meanwhile if some of the requirements changes or new requirements are there in this phase, then one can move to requirement phase to cope with it. Similarly, if there are some problems with requirements in the coding and implementation phase then one can move to the requirement phase from it. The same fashion is followed for the remaining phases. One can move from any phase to any other phase, i.e. from operational requirements to operational design and vice versa, from operational design to coding and implementation phase and vice versa, also from coding and implementation to operational requirements. This property makes it different from other models. The detail of each phase is as under:

A. Operational Requirements:

The requirements are divided into two main categories, functional requirements, and non-functional requirements. The functional requirements describe the core functionality which the system should provide. The non-functional requirements are other requirements like reliability, usability, scalability etc. But the point of focusing is functional requirements because the software is more concerned with the functional requirements. The main methodology of this model is dividing the things and conquering them. Now the functional requirements are further divided in two categories dependent and independent requirements. The requirements at this stage are completely divided and are easy to understand and there are very less chance of being missed. Another important aspect of this model is that the customer is involved in this phase i.e. customer has to verify the requirements to make it sure that developers are going in the right direction. Next step is to analyze the risk for dependent and independent requirements as well as for non-functional requirements. Also risk resolution and management is the parallel activity in this part. In this step we have to find the areas of uncertainty that are the source of risk, and developing a cost effective strategy to resolve that risk.

Then all the requirements i.e. dependent, independent and non-functional requirements are tested separately and in parallel. Here, testing is not a separate phase in this model but testing is involved in each and every phase, which allows finding the bugs earlier that reduces the cost. After testing and risk analysis the divided requirements are integrated and integration testing is done to check whether the requirements are integrated correctly or not. Now the integrated requirements are validated by the customers to cope with the incorrect requirements anomalies. The output of this phase is complete, correct and under-stable objective driven requirements. They are called as objective driven requirements, because they gives us the overall goal and objectives of the product to be developed.

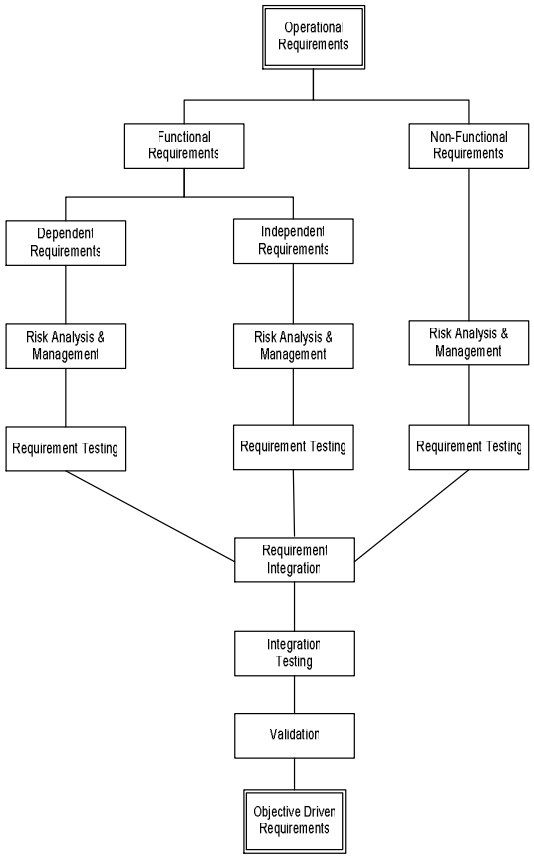

Fig. 2 Operational Requirements

B. Operational Design:

In this phase requirements are divided in the same manner as in the previous phase i.e. in independent and dependent requirements. Here once the requirements are divided now its time to make the initial design from the divided requirements. The design will be much easier and simple because we have

divided the requirements and it will be very easy to carry out. Then the design is validated by the user to reduce the risk of getting into incorrect and bad design. After all these activities initial designs are combined to create single overall design also the prototype is produced to get it validated. If, this prototype is operationally useful and robust enough to serve as a low risk for future product evolution, then this prototype is given to the user for validation. Testing the different designs at this stage is very important to find out the bugs as many as possible. Risk analysis is also very important to find out the future problems, so there should be some plan to manage the risk.

Now the overall design of the system should be validate by the customer. Customer involvement is very important, it will refine the design correctly, doesn't matter whether the customer is technical or not. Customer can understand the design at abstract level. The output of the phase is objective driven design which is passed as an input to the next phase.

C. Coding & Implementation:

The dependent and independent design components are then extracted by splitting the objective driven design. The important advantage of this phase is that the design components are separated which makes it easy to implement them as well as individual designs are easy to handle. Different designs are now easy to test here the unit testing is involved. Unit testing is type of testing in which individual components of the system are tested. After this code integration is done to get the overall product. The integration testing is involved at this part to test the integrated code. Then overall risk analysis and management activities take place which is followed by overall system testing to check the functionality of the system and needs of the customer. Here some other types of testing like usability testing is also done to cope with all needs. Then system is verified and validated. The out put of this phase is complete tested product for deployment.

IV. Edge over other Models:

Waterfall assumes that the requirement can be frozen before the design begins. It is not possible for new system. In divide and conquer model, we haven't frozen requirement phase, one can jump easily from design phase to requirement phase as there is any change in requirement or there is any circumstance in which there is any need to change requirement. Also there is no problem if technology changes. It can be easily handled by going to the requirement phase. We can easily incorporate small to large changes in any phase.

Evolutionary Development divides the development life cycle into increments, and at the end of life cycle user will be able to access the product. The model includes the excessive user involvement, this drawback is coped with the idea in the model that user is not involved at every step, only there is user involvement whenever required.

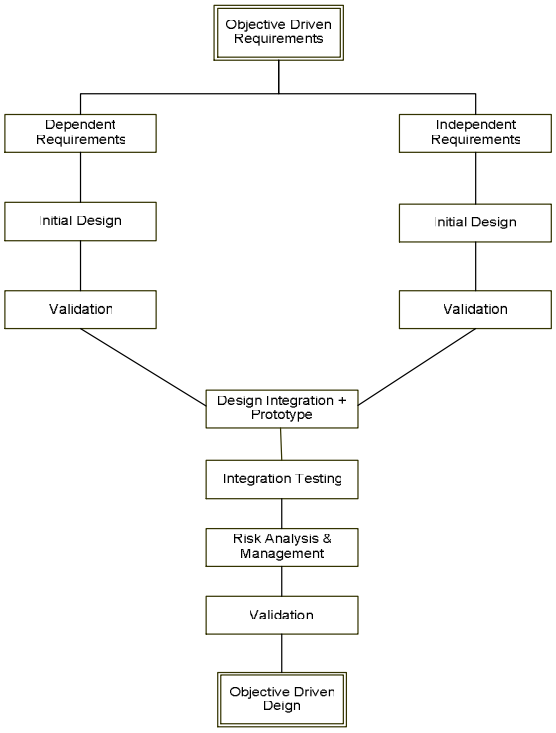

Fig. 3 Operational Design

In prototype model, the prototype is developed. It is based on the idea, that the development continues with the feedback from the prototype is received, and the prototype is evaluated by user and is reworked until customers is satisfied which leads to the disadvantage that it is not known in advance how long it will take to complete a project. In divide and conquer model, prototype is created only in the design phase when design is integrated to give the feel and the look to the user. In prototype model, premature prototype lack key considerations like security, fault tolerance, reliability etc. We have separated non functional requirement from functional requirements to give them special emphasis and considerations so to have strong architecture later on.

V.CONCLUSION

By clearly analyzing the Divide and conquer process model, we have concluded that model is very good for large software projects. It has clear edge over the models like waterfall, prototype and evolutionary development. The model emphasizes on looping back to any phase, to cope the changes. So that we can have a required product, in required time and budget. Most of the process models have increased cost factor due to the fact, that they froze the phase and can't go back to that phase, which lead to incorrect software functionality and high cost. We have proposed the solution which doesn't froze any phase and we can jump to any phase whenever required.

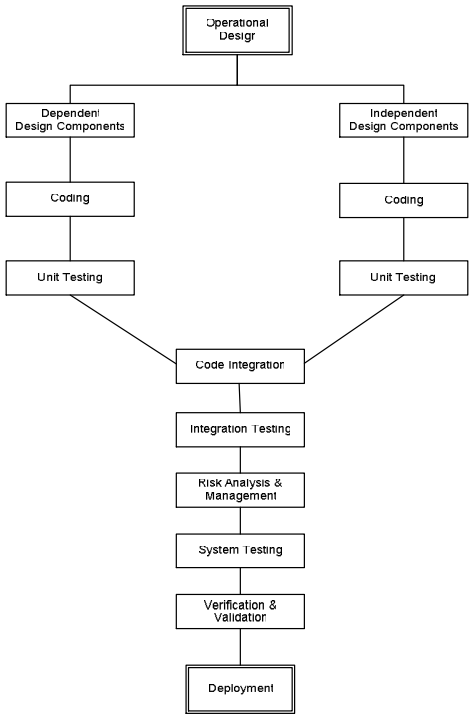

Fig. 4 Coding and Implementation

REFERENCES

[1] Software_Development_Life_Cycle (2009), Software Development Life Cycle. http://en.wikipedia.org/wiki/Software_Development_Life_Cycle, Accessed, September 30, 2009.

[2] R.S. Pressman, "*Software Engineering, A Practitioner's Approach*", 5th ed. New York: McGraw-Hill, 2001, pp. 26.

[3] B.W. Boehm, "*A Spiral Model for Software Development and Enhancement*", IEEE, IEEE Computer Society, vol. 21, issue 5, May 1988, pp. 61 - 72

[4] W.W. Royce, "*Managing the Development of Large Software Systems: Concepts and Techniques*", IEEE, IEEE Computer Society, August 1970, pp. 1-9.

[5] R.J. Madachy, "*Software Process Dynamics*", New Jersey: Willey Interscience, 2007, pp. 32.

[6] R.J. Madachy, "*Software Process Dynamics*", New Jersey: Willey Interscience, 2007, pp. 31.

[7] R.S. Pressman, "*Software Engineering, A Practitioner's Approach*", 5th ed. New York: McGraw-Hill, 2001, pp. 32.

[8] E. Carmel, S. Becker, "*A Process Model for Packaged Software Development*", IEEE, IEEE Transaction on Engineering Management, vol. 42, Feb 1995, pp. 50-58.

[9] E.I. May, B. A. Zimmer, "*The Evolutionary Development Model for Software*", Hewlett-Packard Journal, Article 4, August 1996, pp. 1-8.

[10] B.W. Boehm, "*Anchoring the Software Process*", IEEE, IEEE Software, vol. 13, issue 4, July 1996, pp. 73-83

[11] R.J. Madachy, "*Software Process Dynamics*", New Jersey: Willey Interscience, 2007, pp. 33.

[12] R.S. Pressman, "*Software Engineering, A Practitioner's Approach*", 5th ed. New York: McGraw-Hill, 2001, pp. 34.

[13] P. Kruchten, *"Rational Unified Process Best Practices for Software Development Teams"*, Canada: rational Software, 2001.