

A SOFTWARE DEVELOPMENT LIFE CYCLE MODEL FOR LOW MAINTENANCE AND CONCURRENCY

Uma Maheswar Reddy C.P¹, Chandra Sekhar R.V.P¹, A.K.Rao² and K.Devsen²

¹Bharatiya Vidya Bhavans Vivekananda P.G College, Secunderabad, India, Osmania University,

²Satyam School of Applied Information Systems.

{Umamaheshwarreddy, chandrasekhar_rvp}@indiatimes.com

{AK_Rao, Devsen_Kruthiventi}@satyam.com

ABSTRACT

MODULAR-MODEL aims at reducing the uncertainty and complexity of project by dividing the project into modules and each logical module is viewed independently so that parallelism is achieved and thus improves efficiency. This model clearly defines the task of complete application including the client and it also incorporates iteration increment, which ensures that the application meets the client's requirement by proper verification and validation, incorporates the demand for faster delivery there-by focuses on value and ROI. Features like Prototyping, Modular-division, Risk Analysis and Clearly Defined Tasks enhance the capability of the model to achieve the planned target with-in the prearranged limits of time, budget and scope. The motivation of the article was the statistics compiled by the Standish Group [1].

DESCRIPTION

The MODULAR MODEL for the software development lifecycle basically divides the complete application into Various modules based on the client's requirements and specifications that are known using prototyping phenomenon. The model typically divides the whole processes into two segments (See figure 1).

- Client/Programmer
- Developer/Testing.

To make the understanding better, the modular model has been further divided into four quadrants (See figure 1).

- I: Developer Phase Involving Client,
- II: Developer Phase Involving Programmer,
- III: Testing Phase Involving Programmer,
- IV: Testing Phase Involving Client.

Q-I: Developer Phase Involving Client

The development process usually begins with the *user requirements* phase. URD of the user requirements is taken as input for the *specification* phase. The specification document will include the inputs to the product and the required outputs. The *specifications* of the client are obtained using *prototyping*. They both then discuss, agreeing on the enhancements and amendments. This cycle of inspection-discussion-amendment is usually repeated until the client's requirements are met. A variety of difficulties can arise during the specification phase. All kinds of risk factors should be avoided by proper risk analysis. After this the specifications are divided into a possible number of logically independent system modules (System Design & Prototyping-1---System Design & Prototyping-N). The specifications of a product spell out *what* the product is to do. The aim of the design phase is to determine *how* the product is to do it. The design team determines the internal structure of the product. Design itself is broken into two parts: **System Design** and **Detail Design**.

In System Design, a description of the product in terms of its modules is given, and the logical independent modules (System Design & Prototyping-1---System Design & Prototyping-N) are divided as far as possible to achieve parallelism. During each system design, algorithms are selected and data structures chosen. The specifications include inputs, outputs and all other external aspects of

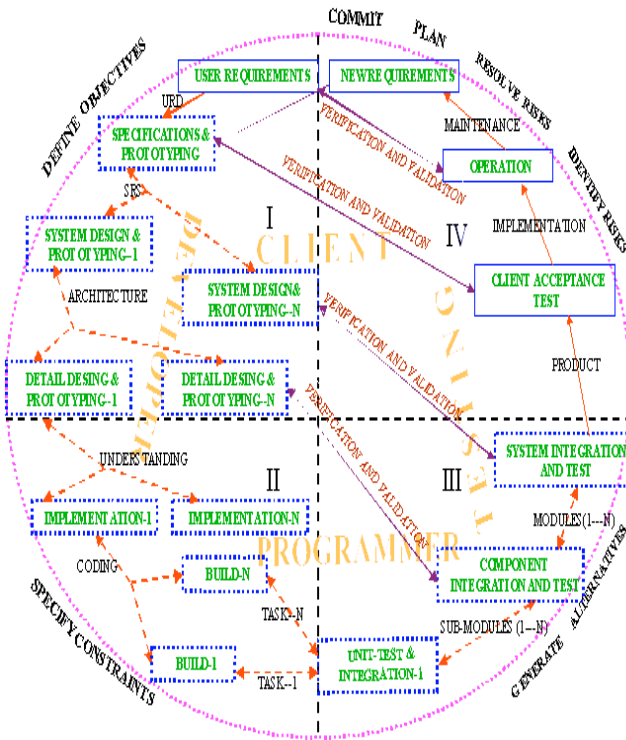


Figure 1: Modular Model

the product. During the design phase, the internal data flows are determined. The design team decomposes the product into *modules*. Each system design (system design & prototyping-1) is future divided into 'n' number of detail design (detail design & prototyping-1---detail design & prototyping-N). The detailed design (detail design & prototyping-1), describes each of the modules and independent pieces of code with well-defined interfaces to the rest of the product. For each module, the designer specifies what that module has to do and how it has to do it. The interface between modules must also be specified in detail. The system design and the detail design both will contain prototype of the product. The prototype thus developed is now verified and validated with an appropriate test. While this is being performed, the design team must keep a careful record of the design decisions that are being made.

Q-II: Developer Phase Involving Programmer

During the implementation phase, the programmer thoroughly understands the detail design and various component modules of the design (Detail Design & Prototyping-1) are coded by dividing the implementation (Implementation-1---Implementation-N) into builds (Build-1---Build-N). The major documentation associated with implementation is the source code with suitable comments.

Q-III: Testing Phase Involving Programmer

All builds obtained from the corresponding implementation undergo *Unit Testing and Integration* (Unit-Test & Integration-1) by the testing team. If the test fails then the (system requirements, rapid prototyping, specification, system design, implementation and build) corresponding process is re-iterated until it succeeds, resulting in a *sub-module*. Similarly from the 'n' implementations we get 'n' sub-modules (Sub-Modules (1---N)). Sub-modules (Sub-Modules (1---N)) thus obtained are *component integrated and tested* resulting in 'n' modules (Modules (1---N)) and then the *system integration test* is performed which integrates all the modules that are divided functionally, resulting in a product. Also simultaneous *verification and validations* are done at the corresponding (component integration test, System integration test, client acceptance test and operation) stages. A comparison with the prototype is performed for every stage mentioned above for better client's acceptance.

Q-IV: Testing Phase Involving Client

This quadrant deals with client *acceptance test* through which client judges the performance of the delivered product in order to meet his needs. If any new requirements are found then again the SRS is written and the whole process is repeated.

ANALYZING WITH EXISTING MODELS

Different models emphasize different aspects of the life cycle, and no single model is appropriate for all software products till date [2]. The more accepted model is the *Waterfall Model*, which includes iteration and feedback after each phase, but the delivered products may not meet client's need. While the *Rapid Prototype Model* ensures that client's requirements are met but the requirement analysis is too little. In *V-Model* each development activity is paired with its corresponding validation activity but does not support iteration while the *Spiral-Model* integrates the risk management with almost all the features of above models, but relies more on developer's experience. *Incremental-Model* delivers operational products within a short span but require open architecture. *Sawtooth* and *W Models* solve the discrepancy between the client and developer but changes in requirement analysis are too difficult to implement. *RUP Model* treats a problem as a collection of objects but may degenerate into *CABTAB* problem [3].

ADVANTAGES

- It Adapts the Divide and Conquer Policy.
- Rational Incorporation of Prototypes
- This Model Solves the Discrepancy by Showing the Users and Software developers perceptions of the system as different trajectories.
- Validation and Verification after Each Step.
- Ensures Delivered Product Meets Client's Needs.
- Integration of Risk Analysis.
- Operational Product Delivered In Short Span.
- Maximizes Early Return On Investments.
- Supports Iterative Incrementation.

CONCLUSION

This model achieves parallelism, avoids the risk involved in the project and client requirements are well understood from the beginning, there-by minimizing the maintenance cost. The highest priority module is identified and delivered first there-by maximizing the *early return on investment*. Thus on the whole this model is most reliable for overall Application Development Processes.

REFERENCES

1. Standish group reports on failure of IT projects (A technical reports [KSI-TN -100103]).
2. Richard-Farley (1985), Software Engineering Concepts, Tata McGraw-Hill.
3. Roger S. Pressman (1994), Software Engineering, Tata McGraw-Hill Edition.