International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 4, April 2014)

Software Engineering Process Models & Trends

Gangadhar.G.H
Asst.Prof ,Dept Of CSE
BKEC, Basavakalyan
gangadharah@yahoo.com

Sneha.M
Asst.Prof ,Dept Of CSE
BKEC, Basavakalyan
mankare.sneha@gmail.com

Shivaraj P Patil
Professor & HOD, Dept of CSE
BKEC, Basavakalyan
patilshivaraj16@gmail.com

Abstract— **Software is an important aspect of modern society. It is seen in every day to day human usable technology from buying bread, driving car, washing clothes to managing finances, communication regulation and generation of power and processing of information in secured manner. Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. The people who develop them are software engineers, s/w developers and s/w programmers. S/W separated from hardware in early 1950s and emerged as distinct and independent technology. Following are s/w development practices but it may vary depending on the process:**

- **Requirements engineering**
- **System analysis**
- **High-level design/architecture**
- **Low-level design**
- **Coding**
- **Integration**
- **Design and code reviews**
- **Testing**
- **Maintenance**
- **Project management**
- **Configuration management**

Index Terms—**Deductive verification, scalable spiral model, automatic programming, agile and plan driven methods, Software verification**

I. INTRODUCTION

A software development process is the process by which user needs are translated into a software product. The process involves translating user needs
into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes installing and checking out the software for operational use. A software process model is a simplified, abstracted description of a software development process. The primary purpose of a software process model is to determine the order of stages involved in software development and to establish the transition criteria for progressing from one stage to the next (Boehm, May 1988).

Process models have begun to be characterized as plan-driven or agile. The plan-driven models have an implicit assumption that a good deal of information about requirements can be obtained up front and that information is fairly stable. Plan-driven models are also considered more suitable for safety- and mission-critical systems because of their emphasis on defect prevention and elimination. Some examples of plan-driven methodologies are the Personal Software Process, the Rational Unified Process,and Cleanroom Software Engineering, agile models are considered to be better suited for projects in which a great deal of change is anticipated. Some examples of agile methodologies are the Extreme Programming (XP) (Beck, 2000), FDD

Software Intensive Systems (SIS): Software intensive systems (SIS) have become the foundation of virtually every modern technology.

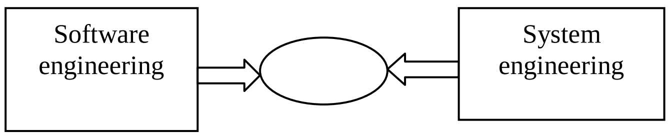


Fig 1:Both s/w and system processes are affecting the system

- Increasing integration of software engineering and system engineering activities
- Increasing emphasis on users and end-value
- Increasing SIS criticality and dependability
- Increasing need to manage rapid change
- Increasing project/product globalization and need for interoperability
- Increasing size and complexity
- Increasing software “autonomy”

Software engineering Process Activities:
Software process means organizing structured set of activities to develop software systems. It involves following activites

- ➢ Specification – defining what the system should do;
- ➢ Design and implementation – defining the organization of the system and implementing the system;
- ➢ Validation – checking that it does what the customer wants;
- ➢ Evolution – changing the system in response to changing customer needs.

Software Process Model:

It is an abstract and simplified description of software development process.It presents a description of a process.Its purpose is to determine the order of stages involved in developmentFollowing are the activities in the process:specifying a data model,designing a user interface and the ordering of these activities,etc.

Types of software processes:

The *plan-driven models* have an implicit assumption that a good deal of information about requirements can be obtained up front and that information is fairly stable. As a result, creating a plan for the project to follow is advisable. A long-standing tenet of software engineering is that the longer a defect remains in a product, the more
expensive it is to remove it. cost of product development can be minimized by creating detailed plans and by constructing and inspecting architecture and design documents. As a result of these activities, there will be significant cost savings because defects will be removed or prevented. Plan-driven models can be summarized as "Do it right the first time." These models are very appropriate for projects in which there is not
a great deal of requirements and/or technology changes anticipated throughout the development cycle. Plan-driven models are also considered more suitable for safety- and mission-critical systems because of their emphasis on defect prevention and elimination.

The agile *models* are considered to be better suited for projects in which a great deal of change is anticipated. Because of the inevitable change, creating a detailed plan would not be worthwhile because it will only change. Spending significant amounts of time creating and inspecting an architecture and detailed design for the whole project is similarly not advisable; it will only change as well. The methodologies of the agile model focus on spending a limited amount of time on planning and requirements gathering early in the process and much more time planning and gathering requirements for small iterations throughout the entire lifecycle of the
project.

Types of Software process models:

- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.

Waterfall model

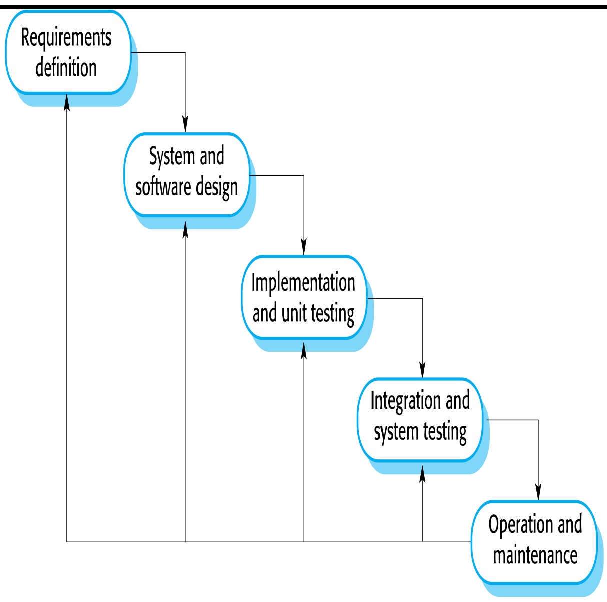
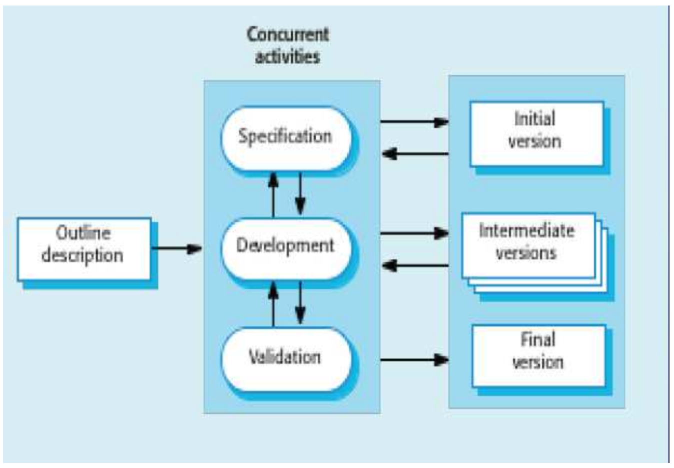

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Fig: Waterfall model

- There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance

Evolutionary development:

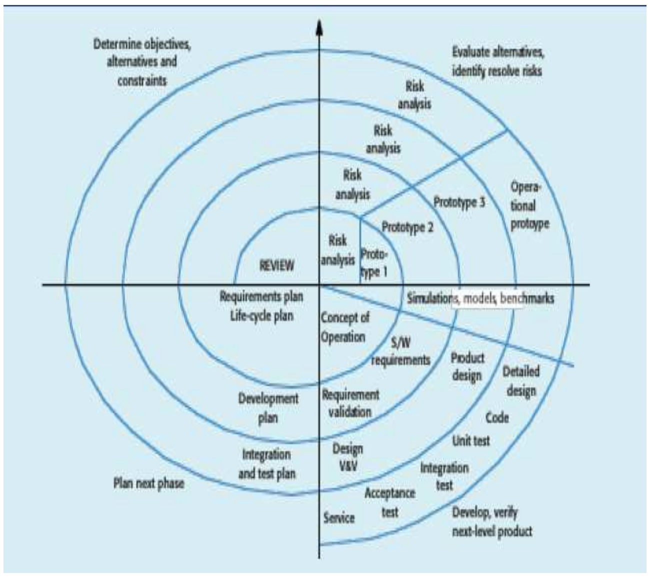
Its objective is to work with customers and to evolve a final system from an initial outline specification.It should start with well understood requirements and add new features as mentioned by customers



Process Iteration:

- Incremental development
- Spiral modeL

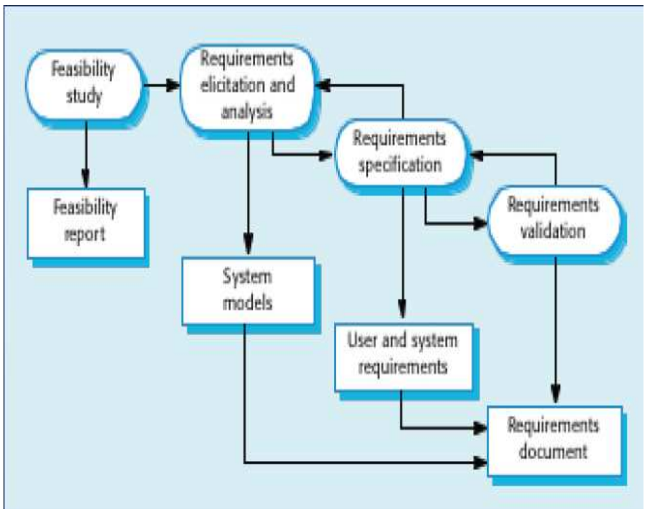
Spiral model: Process is represented as spiral rather than sequence of activities with backtracking.Each loop in the spiral represent a phase in the process.



Software Specification: The process of establishing what services are required and the constraints on the system's operation and development.

1.Requirements Engineering

- To improve the manner in which requirements aredefined, the software engineering community will likely implement three distinct sub-processes as RE is conducted [Gli07]
- improved knowledge acquisition and knowledge sharing that allows more complete understanding of application domain constraints and stakeholder needs
- greater emphasis on iteration as requirements are defined more effective communication and coordination tools that enable all stakeholders to collaborate effectively



II. **Software Engineering Trends:**

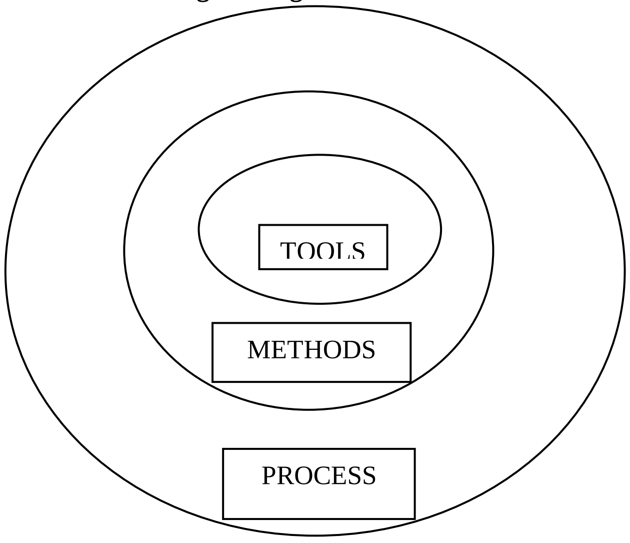


Fig 5: Trends in Software engg

a.SE Trends—Process:

Agility
Adaptability
Collaboration
Communication
ROI

b.SE Trends—Methods:

In RE
In design
In languages
In testing
In SCM/SQA

c.SE Trends—Tools:

Project mgmt.
Modeling
Programming
Testing
Maintenance
SCM
SEEs

The growing development of dependable software has its own importance. Formal methods in s/w engg is an emerging application of intelligent systems. Deductive s/w verification is a vital technology in formal techniques. Subsequent to this, is s/w synthesizing which is an important base technique of formal methods. Software verification: It is a formal technique for reasoning about properties of programs. It uses computer assistance for proof search book keeping. It proves validity by deduction in logic calculususe.Verified s/w consists of programs that are proved error free. Deductive s/w verification is a formal technique for reasoning about properties of programs that has been around 40 years. Earlier verification of algorithms in academic languages was done, now verification tools support programming languages like java, C#. Earlier, verification tools used to be stand alone applications used after

years of training. Now, tools are there requiring minimal training to use. Most importantly, deductive verification tools are used as base technology in formal s/w verification and automatic test generation as well. In contrast to static analysis and model checking, can model the semantics of target programming without abstracting from unbounded data structures or programming constructs. The logic In deductive verification are least expensive so it can be formalized and proved far reaching target properties. A recent trend is to insist on automation but to accept approximate results contrast with static analysis and model checking. In logical modeling of programming languages and their semantics, two approaches exist. In the first, target programs appear as separate syntactical category embedded in logical expression. Hoare logic is the best known. Logical rules that characterize the derivability of those formulas that contain programs reflect the target programming language's operational semantics. In the second, target language syntax and semantics are encoded as theories. It involves formalizing data structures like sets, functions, lists, tuples and records.
One major trend is that verification tools attempt to combine formal verification, automatic test generation and automatic bug finding into a uniform framework that is integrated into standard software development platforms like eclipse. The forces behind them are formal verification is too limited and they are at the source code.

TRENDS:
1. System engineering and software engineering integrated together
2. Rapid change
3. Increased complex systems of systems
4. More demand for COTS, reuse and legacy SIS integration
5. Need for dependability and increased SIS criticality
Many trends have been the reason for evolvement of system engineering and software engineering as large sequential and independent processes systems like ships, railroad

1.Firstly sys eng begun to determine configuring of various h/w components into physical systems like ships, railroads or defense systems. After this functional components and information requirements were specified then external or internal contracts are defined in sequence to produce the components. Secondly, s/w engineering is influenced by a highly formal and mathematical approaches for specifying s/w components and a reductionalist approach for deriving computer s/w programs that correctly implemented formal specifications. Thirdly, in places like government sectors, specification & standards were well placed and were difficult change so "purchasing agent" concept was followed wherein requirements were sequentially specified, contracts were established, formulated and implementing of solutions was done and requirements were used for acceptance-test solutions.

2. User-or-organization desire is to have technology that adopts to people rather than vice0versa.Enterprise support packages, data access and mining tools and PDAs are technologies affecting usability and cost effective challenges of this trend is requirements are not prespecified but emerged which are not compatible with past process like requirement driven sequential water fall process models and formal programming calculi.

3. Other challenge is that dependability is not top priority for s/w products .scaling up and integrating s/w and system dependability so that they cope with future trends challenges like rapid change and agility, globalization, complex system of system and COTS/legacy integration is big hurdle. Examples are AT&T telephone n/w's where high dependability is achieved.

4. Rapid change increases the priority of developing speed vs cost in capitalizing on market window .Hewlett Packard's initiative is one of the example to reduce product line s/w development times from 48 to 12 months, change is driven by trends like Gordan Moore's law plus continuing need for product difference and global connectivity accelerates ripple effects of technology, market place and technology changes.

5. Globalization provides major economies of scale and n/w economies that drive organizations product and process strategies. A standard based infrastructure is essential for effective global collaboration; example is low adoption rate of mire individual/masculine/short term US culture's s/w CMM by organizations in the more collective/feminine/long term that culture.

6. System and software development processes have high risks of inadequate interoperability with other systems. Software intensive systems risks have following trends: acquisition management and staffing, requirements/architecture feasibility, achievable software schedules, supplier integration, adoption to rapid change, system and s/w quality factor achievability, product integration and electronic upgrade. To keep software intensive systems of systems destabilized from change of traffic, organize development into plan driven increments(to keep stable from changes).

7.Infrastructure S/w developers continue to spend most of their time programming and application s/w developers spend in assessing, tailoring and integrating commercial off the shelf (COTS) products. These are challenging to integrate, as they are hard to debug and incompatible with each other. Open source s/w (organization's reused or legacy software) is less opaque. But this to have problems with interoperability and continuing evolution. It put constraints on new applications incremental development. COTS, open source, reused, and legacy s/w and h/w will have shortfalls in dependability and interoperability. IT can be overcome by increasing customer and enterprise architecture initiatives .

8.In 21st century system and s/w development and evolution, The common modes are business model based user

programming, hardware and software product lines, development of unprecedented capabilities, network centric systems of systems. Exploratory development processes will continue to be used in mainstream organizations and in new areas like nanotechnology, advanced biotechnology and robotics, virtual reality and co-operative agent based systems.
Business model based user programming address the need to produce more and more s/w capabilities by enabling them to be produced directly by users like spread sheet programs, computer aided design and manufacturing and website development and evolution. Hardware and software product lines includes product lines for transportation, communications, medical, construction, business services, public services and information infrastructure. Network centric systems of systems are highly software intensive and need to be robust, scalable and evolvable in flexible.

III. Software Engineering Challenges:

• *Tractable Medium.*
Quite often programmers are also asked to fix hardware product problems because people think that it is cheaper to fix the problems in the (tractable) software than it is to re-design and re-manufacture physical parts. This presents software engineers with the need to design and coding changes, often at the last minute. The software industry has been trying to formulate a sort of scientific/mathematical basis for itself. Formal notations have been proposed to specify a program; mathematical proofs have been defined using these formal notations. The software community is also establishing analysis and design
patterns.

• *Changing requirements*. Adapting for hardware changes is only one source of requirements churn for software engineers. Unfortunately, requirements changes come from many sources. It is often very hard for customers to express exactly what they want in a product .Requirements analysts may not understand the product domain as completely as they need to early in the product lifecycle. As a result, the analysts might not know the right questions to ask the customer to elicit all their requirements .Lastly; the product domain can be constantly changing during the course of a product development cycle. New technology becomes available. Competitors release new products that have features that weren't thought of. Innovators think
of wonderful new ideas that will make the product more competitive.

• *Schedule Optimism*. Software engineers are an optimistic crew. In most organizations, it is the software engineers who estimate how long it will take to develop a product. No matter how many times we've taken longer than we thought in the past, we still believe "Next time, things will go more smoothly. We know so much more now"

• *Schedule Pressure*. We often make these aggressive commitments because of the
intensity of the people asking us for commitment. It seems that every product is late before it's even started, every feature is critical or the business will fold. Products need to be created and updated at a constant, rapid pace lest competitors
take over the business.

CONCLUSION: In this paper we emphasized on recent trends and methods deployed in software engineering streams, by taking into account deductive verification, enhanced spiral model and automatic programming. Future works can be done on improving the testing practice on emerging spiral models and also putting efforts in synchronizing system and software processes. Challenges faced by software product development like varying requirements, Scheduled pressure and scheduled optimism can be worken on it.

REFERENCES

[1]. Boehm, A spiral model for software development and enhancement, Computer (May 1988), 61–72.
[2]. Beck, Extreme programming explained, Addison Wesley, Reading, MA, 1999.
[3]. A. Fuggetta, "Software process: A roadmap," The future ofsoftware engineering, A. Finkelstein (Editor), ACM Press,2000.
[4]. L. Huang, A value-based process achieving software dependability,Proc Software Process, Workshop 2005, May 2005.
[5]. M. Paulk, C. Weber, B. Curtis, and M. Chrissis, The Capability Maturity Model, Addison Wesley, Reading, MA, 1994.
[6]. Y. Yang, B. Boehm, and D. Port, A contextualized study of COTS-based e-service projects, Proc Software Process Workshop 2005, Springer, New York, 2005,
[7]. Y. Yang, J. Bhuta, D. Port, and B. Boehm, Value-based processes for COTS-based applications, IEEE Software (July/August 2005), 54–62.
[8]. Intelligent systems and formal methods in software engineering,by Bernhard Beckert.
[9]. B. Boehm, A.W. Brown, V. Basili, and R. Turner, Spiral acquisition of software-intensive systems of systems, CrossTalk (May 2004), 4–9.