International Journal of Innovations in Computing
(ISSN : 2319-8257) Vol. 2 Issue 3

Current Trends in Software Engineering

Raghvendra Singh
MCS College of Engg. & Tech., Lucknow
raghvendra711@gmail.com

Abstract— ***Software Engineering come and go through a series of passages that account for their inception, initial development, productive operation, upkeep, and retirement from one generation to another. This paper categorizes and examines a number of methods for describing or modeling how software systems are developed. In this paper we are describing that the contemporary models of software development must account for software the interrelationships between software products and production processes, as well as for the roles played by tools, people and their workplaces.***

Keywords- Software Engineering, Software Project, Software life cycle models.

I. INTRODUCTION

Software life cycle models are followed by a more comprehensive review of the alternative models of software evolution that are of current use as the basis for organizing software engineering projects and technologies. A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed.

A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes or for building empirically grounded prescriptive models.

A prescriptive model prescribes how a new software system should be developed. Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed, and in what order. Typically, it is easier and more common to articulate a prescriptive life cycle model for how software systems should be developed. This is possible since most such models are intuitive or well reasoned. This means that many idiosyncratic details that describe how a software system is built in practice can be ignored, generalized, or deferred for later consideration.

This, of course, should raise concern for the relative validity and robustness of such life cycle models when developing different kinds of application systems, in different kinds of development settings, using different programming languages, with differentially skilled staff, etc. However, prescriptive models are also used to package the development tasks and techniques for using a given set of software engineering tools or environment during a development project.

These classic software life cycle models usually include some version or subset of the following activities:

- System Initiation/Planning
- Requirement Analysis and Specification
- Functional Specification or Prototyping
- Partition
- Architectural Design and Configuration Specification
- Detailed Component Design Specification
- Component Implementation and Debugging
- Software Integration and Testing
- Documentation Revision and System Delivery
- Deployment and Installation
- Training and Use
- Software Maintenance

II. SOFTWARE PROCESS NETWORKS

It can be viewed as representing multiple interconnected task chains .Task chains represent a non-linear sequence of actions that structure and transform available computational objects (resources) into intermediate or finished products.

Non-linearity implies that the sequence of actions may be non-deterministic, iterative, accommodate multiple/parallel alternatives, as well as partially ordered to account for incremental progress. Task chains can be employed to characterize either prescriptive or descriptive action sequences. Prescriptive task chains are idealized plans of what actions should be accomplished, and in what order. For example, a task chain for the activity of object-oriented software design might include the following task actions:

- Develop an informal narrative specification of the system.
- Identify the objects and their attributes.
- Identify the operations on the objects.
- Identify the interfaces between objects, attributes, or operations.
- Implement the operations.

Clearly, this sequence of actions could entail multiple iterations and non-procedural primitive action invocations in the course of incrementally progressing toward an object-oriented software design.

III. SOFTWARE PROCESS EVOLUTION

Based on the literature, we can identify five properties that characterize the evolution of large software systems. These are:

- *Continuing change*: a large software system undergoes continuing change or becomes progressively less useful.
- *Increasing complexity*: as a software system evolves, its complexity increases unless work is done to maintain or reduce it.
- *Fundamental law of program evolution*: program evolution, the programming process, and global

measures of project and system attributes are statistically self-regulating with determinable trends and invariance's.

- *Invariant work rate*: the rate of global activity in a large software project is statistically invariant
- *Incremental growth limit*: during the active life of a large program, the volume of modifications made to successive releases is statistically invariant.

IV. SOFTWARE TECHNOLOGY MATURATION

Software engineering will benefit from a better understanding of the research strategies that have been most successful. The model presented here reflects the character of the discipline: it identifies the types of questions software engineers find interesting, the types of results we produce in answering those questions, and the types of evidence that we use to evaluate the results.

It is found that it typically takes 15-20 years for a technology to evolve from concept formulation to the point where it's ready for popularization. There are six typical phases:

A. Basic research.

Investigate basic ideas and concepts, put initial structure on the problem, frame critical research questions.

B. Concept formulation.

Circulate ideas informally, develop a research community, converge on a compatible set of ideas, publish solutions to specific subproblems.

C. Development and extension.

Make preliminary use of the technology, clarify underlying ideas, generalize the approach.

D. Internal enhancement and exploration.

Extend approach to another domain, use technology for real problems, stabilize technology, develop training materials, show value in results.

E. External enhancement and exploration.

Similar to internal, but involving a broader community of people who weren't developers, show substantial evidence of value and applicability.

F. Popularization.

Develop production-quality, supported versions of the technology, commercialize and market technology, expand user community.

V. TYPE OF RESULT EXAMPLES AND PROCEDURE

New or better way to do some task, such as design, implementation, maintenance, measurement, evaluation, selection from alternatives; includes techniques for implementation, representation, management, and analysis; a technique should be operational—not advice or guidelines, but a procedure.

A. Qualitative or descriptive model

Structure or taxonomy for a problem area; architectural style, framework, or design pattern; non-formal domain analysis, well-grounded checklists, well-argued informal generalizations, guidance for integrating other results, well-organized interesting observations

B. Empirical model

Empirical predictive model based on observed data

C. Analytic model

Structural model that permits formal analysis or automatic manipulation

D. Tool or notation

Implemented tool that embodies a technique; formal language to support a technique or model (should have a calculus, semantics, or other basis for computing or doing inference)

E. Specific solution, prototype, answer, or judgment

Solution to application problem that shows application of SE principles – may be design, prototype, or full implementation; careful analysis of a system or its development, result of a specific analysis, evaluation, or comparison

F. Report

Interesting observations, rules of thumb, but not sufficiently general or systematic to rise to the level of a descriptive model.

VI. CURRENT TRENDS AND NEW DIRECTIONS

Software engineering does not have this sort of well-understood guidance. Software engineering researchers rarely write explicitly about their paradigms of research and their standards for judging quality of results. A number of attempts to characterize software engineering research have contributed elements of the answer, but they do not yet paint a comprehensive picture.

Research questions are of different kinds, and research strategies vary in response. The strategy of a research project should select a result, an approach to obtaining the result, and a validation strategy appropriate to the research question. The questions of interest change as the field matures. One indication that ideas are maturing is a shift from qualitative and empirical understanding to precise and quantitative models.

More recently, software engineering researchers have criticized common practice in the field for failing to collect, analyze, and report experimental measurements in research reports presented preliminary sketches of some of the successful paradigms for software engineering research, drawing heavily on examples from software architecture.

Scientific and engineering research fields can be characterized by identifying what they value:

- What kinds of questions are "interesting"?
- What kinds of results help to answer these questions, and what research methods can produce these results?
- What kinds of evidence can demonstrate the validity of a result, and how are good results distinguished from bad ones?

In this paper we attempted to make generally accepted research strategies in software engineering explicit

by examining research in the area to identify what is widely accepted in practice.

These opportunities areas and sample direction for further exploration include:

- Software process simulation
- Web-based software process models and process engineering
- Software process and business process reengineering
- Understanding, capturing, and operationalizing process models

VII. CONCLUSION

Many sciences have good explanations of their research strategies. These explanations include not only detailed guidance for researchers but also simplified views for the public and other observers. In addition to the ongoing interest, debate, and assessment of process-centered or process-driven software engineering environments that rely on process models to configure or control their operation, there are a number of promising avenues for further research and development with software process models.

Nonetheless, we must also recognize that the death of the traditional system life cycle model may be at hand. New models for software development enabled by the Internet, group facilitation and distant coordination within open source software communities, and shifting business imperatives in response to these conditions are giving rise to a new generation of software processes and process models. These new models provide a view of software development and evolution that is incremental, iterative, ongoing, interactive, and sensitive to social and organizational circumstances, while at the same time, increasingly amenable to automated support, facilitation, and collaboration over the distances of space and time.

REFERENCES

[1] Victor R. Basili. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directives*. Proc of Dagstuhl- Workshop, H. Dieter Rombach, Victor R. Basili, and Richard Selby (eds), published as *Lecture Notes in Computer Science #706*, Springer-Verlag 1993.

[2] Geoffrey Bowker and Susan Leigh Star: *Sorting Things Out: Classification and Its Consequences*. MIT Press, 1999

[3] Thomas F. Gieryn. *Cultural Boundaries of Science: Credibility on the line*. Univ of Chicago Press, 1999.

[4] Frederick P. Brooks, Jr. Grasping Reality Through Illusion -- Interactive Graphics Serving Science. *Proc 1988 ACM SIGCHI Human Factors in Computer Systems Conference (CHI '88)* pp. 1-11.

[5] Impact Project. "Determining the impact of software engineering research upon practice. Panel summary, *Proc. 23rd International Conference on Software Engineering (ICSE 2001)*, 2001

[6] S. Redwine & W. Riddle. Software technology maturation. *Proceedings of the Eighth International Conference on Software Engineering*, May 1985, pp. 189-200.

[7] William Newman et al. *Guide to Successful Papers Submission at CHI 2001*. http://acm.org/sigs/sigchi/chi2001/call/submissions/guide-papers.html

[8] OOPSLA '91 Program Committee. How to get your paper accepted at OOPSLA. *Proc OOPSLA'91*, pp.359-363.

[9] Craig Partridge. How to Increase the Chances your Paper is Accepted at ACM SIGCOMM. http://www.acm.org/sigcomm/conference-misc/author-guide.html

[10] Samuel Redwine, et al. *DoD Related Software Technology Requirements, Practices, and Prospects for the Future*. IDA Paper P-1788, June 1984.

[11] Marvin V. Zelkowitz and Delores Wallace. Experimental models for validating technology. *IEEE Computer*, Vol. 31, No. 5, 1998, pp.23-31.

[12] Mary Shaw. Prospects for an engineering discipline of software. *IEEE Software*, November 1990, pp. 15-24.

[13] Mary Shaw. The coming-of-age of software architecture research. *Proc. 23rd International Conference on Software Engineering (ICSE 2001)*, pp. 656-664a.

[14] W. F. Tichy, P. Lukowicz, L. Prechelt, & E. A. Heinz. "Experimental evaluation in computer science: A quantitative study." *Journal of Systems Software*, Vol. 28, No. 1, 1995, pp. 9-18.

[15] Walter F. Tichy. "Should computer scientists experiment more? 16 reasons to avoid experimentation." *IEEE Computer*, Vol. 31, No. 5, May 1998

[16] Marvin V. Zelkowitz and Delores Wallace. Experimental validation in software engineering. *Information and Software Technology*, Vol 39, no 11, 1997, pp. 735-744.