ソフトウェア工学 105−11
（1995. 9. 8）

# プロセス・モデリングと高信頼性システム

劉 少 英, 大場 充
shaoying@cs.hiroshima-cu.ac.jp
ohba@cs.hiroshima-cu.ac.jp
広島市立大学, 情報科学部
〒 731-31 広島市安佐南区沼田町大塚 151-5

Yong Sun
y.sun@qub.ac.uk
Department of Computer Science
The Queen's University of Belfast
Belfast BT7 1NN
U.K.

## 要旨

ソフトウェア工学において、形式的な方法の応用は高信頼性システムの開発に非常に有用である。しかし今日まで、形式的な方法に基ずいたプロセスが完成された例はまだ報告されていない。本論文では、プロセス・モデリングの一般的な技法を調べた上で、高信頼性システムを開発するための形式的な方法に基ずいた機能的安全性を保証するインタラクティブ開発プロセスを提案する。

# Process Modelling and Safety Critical Systems

Shaoying Liu and Mitsuru Ohba
Faculty of Information Sciences
Hiroshima City University
151-5, Asaminami-ku, Hiroshima 731-31
Japan

Yong Sun
Department of Computer Science
The Queen's University of Belfast
Belfast BT7 1NN
U.K.

Formal methods have been recognized to be extremely useful for development of safety critical systems. However, there are so far no mature processes for developing safety critical systems using formal methods. In this paper, we investigate most popular techniques in process modelling, and proposes a function-safety interactive development process for developing safety critical systems using formal methods.

## 1 Introduction

Our major concern is how to incorporate formal methods and safety analysis techniques into the software process (i.e. the process of software development) in order to produce a software development methodology for safety critical system. Several software process models exist, such as Waterfall model, Spiral model, and formal development model, but none of them clearly indicates how to carry out both the functional and safety analysis during the development of a safety critical system. Since formal methods have been recognized by the community as a way of increasing confidence in software for safety critical systems, we believe that establishing a methodology for safety-critical systems development based on the formal software development model will facilitate and encourage the application of formal methods in safety-critical systems.

In this paper, we first analysize the existing software process models, and then propose a function-safety interactive development process for developing safety-critical systems using formal methods. At this moment, our ideas reflected by this proposal have not been converted into specific techniques, nor have they been applied to a practical project. In this sense, the work presented in this paper is premature. However, we hope that our proposal will lead to a technical progress in the field.

The remainder of this paper is organized as follows. Section 2 describes several existing software process models. In section 3, we present a function-safety interactive analysis process for development of safety critical systems. Finally, conclusions and future research are given in section 4.

## 2 Software Development Process Models

Several software development process models exist. They include *Waterfall Model*, *Spiral Model*, *Successive Process Model* and *Successive Refinement Model*. The first two have been popular for the development of large software project in industry, while the latter two are only under research. In this section, we review all these four models, respectively.

### 2.1 Waterfall Model

The waterfall model presents a transformational process for software development and maintenance from *System requirements*, through *Software requirements*, *Analysis*, *Program design*, *Coding*, *Testing*, to *Operations and maintenance* [Royce 70], as shown in Figure 1.

Apart from the transformational process in the waterfall model, the interactions between the various phases are encouraged. The intention is that the process remains under control if all rework needs only to go back one step for the team to retrieve the situation and then to have a basis for further progress. This model of the process of working makes good sense when the software is developed by an individual, or by a team acting cohesively as an individual. However, for a large project involving more people, organizations and managements, this process is generally not easily controllable. Design iterations may not be confined to successive stages, leading to the sort of situation that every two stages are likely interactive.

### 2.2 Spiral Model

The spiral model has been developed over a number of years, based on experience on large government software projects [Boehm 88]. It can be used to discuss the principles of processes to fit a wide variety of circumstances and provides guidance as to which sequence of phases best fits a given software situation. Compared with the waterfall model, the spiral model advocates prototyping during the development process and considers cost as an important element, as shown in Figure 2.

There are two dimensions in the spiral model, *radial and angular dimensions.* The radial dimension represents the cumulative cost incurred in accomplishing the steps to date; the angular dimension represents the progress made in completing each cycle of the spiral. The model holds that each cycle involves a progression through

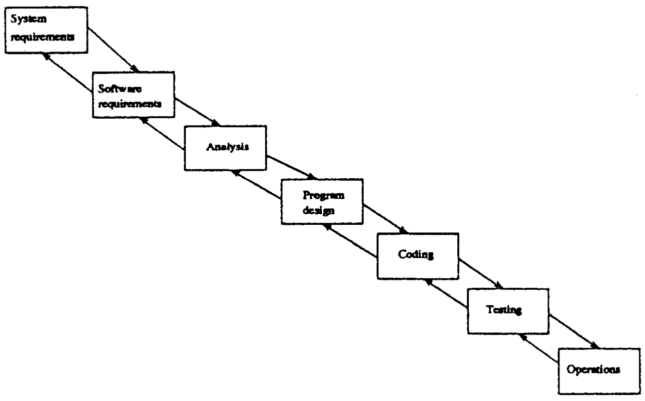

Figure 1: Waterfall model of software development

the same sequence of steps, for each portion of the product and for each of its levels of elaboration, from an overall concept of operation paper down to the coding of each individual program.

The development of a software product within the spiral model starts with the determination of the objectives of the product being elaborated (performance, functionality, ability to accommodate change, etc.), the alternative means of implementing this product, and the constraints imposed on the application of the alternatives (cost, schedule, interface, etc.). Then the evaluation of the alternatives with respect to the objectives and constraints is carried out. This evaluation must identify any areas of uncertainty which are significant sources of project risk, and formulate a cost-effective strategy for dealing with the sources of risk. This may involve prototyping, simulation, questioning users, analytic modelling, or combinations of these and other risk-resolution techniques. Once the risk is evaluated, the next step is determined by the relative balance of the perceived risks and further development is carried out.

### 2.3 Successive Process Model

Although both the waterfall and the spiral models provide the principles for software development and a easy way for project managements, there are many defficiencies. First, the principles only indicate a set of informal guidelines on system development. Second, there is no precise obligations defined for the activities at each development stage. Third, it is difficult in practice to decide the boundaries between different development stages (e.g. when should the requirements analysis end? when should the design stops?) so the project management in fact is more difficult than it is expected. In order to improve the capacity of the existing models, a successive process model is proposed [Salter 1993], as shown in Figure 3.

According to the successive process model, a software development consists of a sequence of software processes, each of which accepting some input information and producing some output information as the input information to a successive process. Processes can be either parallel or sequential during development.

In fact, the successive process model is the result of detailing the waterfall model, but it represents a more general principle for software development. The process of a software development is not divided into several stages but processes which gradually transform the user's requirements into a final software product.

However, the successive process model provide no precise principle of how to divide a development into processes, no precise relationship between successive processes, and no precise obli-

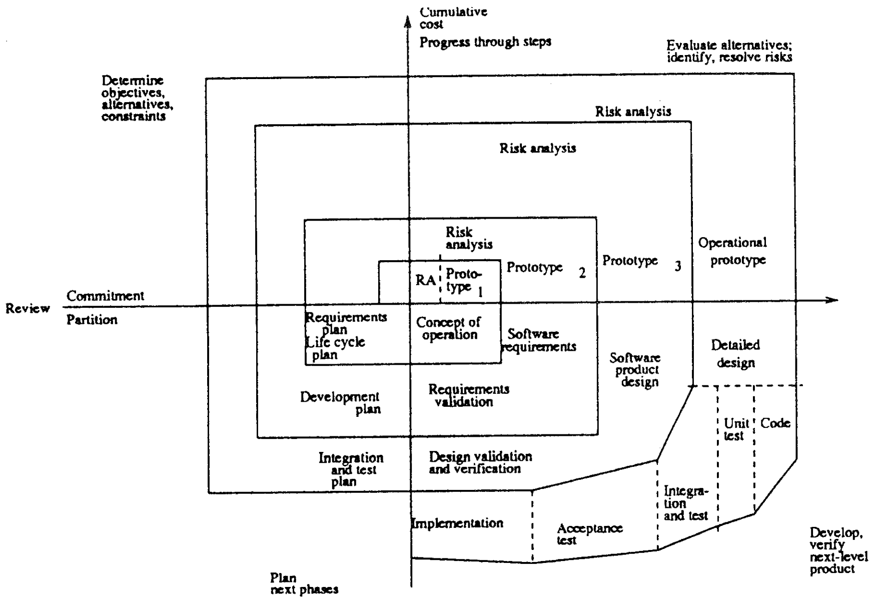

Figure 2: Spiral model of software development

gations for the development of each process. A major reason for the limitation is because the input and output information of each process is not required to be expressed in formal notations.

## 2.4 Successive Refinement Model

The successive refinement model for software development is based on formal methods. According to this model, software development is a process of successive refinements from abstract specifications into concrete specifications, as shown in Figure 4. The top level abstract specification is derived from the user requirements by means of formalisation and its correctness must be checked in agreement with the user through validation. This specification is then transformed into a real system (i.e. program) by a sequence of refinements. Each refinement transforms an abstract specification (which includes more information on *what to do* but less information on *how to do* it) into a concrete specification (which includes more information on *how to do* it). Formal proof demonstrates statically that the refined concrete specification satisfies the corresponding abstract specification, and testing demonstrates dynamically that the program satisfies the user's true requirements.

Based on the model of the software life-cycle the process described in Figure 4 can be understood as follows. The 'User Requirements' is the result of *requirements analysis* and is normally described in informal language (e.g. English). 'Specification$_1$' corresponds to the stage of *functional specification*, and from 'Specification$_2$' to 'Specification$_n$' corresponds to the stage of *design*. The refinement from 'Specification$_n$' into 'Program' corresponds to the stage of *implementation*. If the desired system is safety critical, the process of software development should involve the activity of safety analysis. In principle, safety analysis should be enforced in every stage of the software development process because the specification (or program) produced in every stage expresses the potential functional behaviour of the current system, which must not violate the corresponding safety requirements. Ways of achieving this are not currently

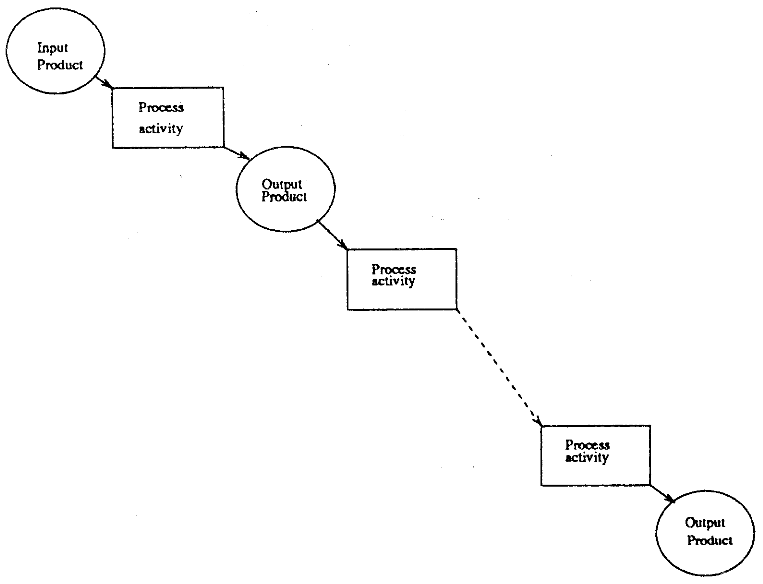

Figure 3: Successive process model of software development

well understood.

Compared with successive process model, the successive refinement model is an improvement in the sense that a precise relationship between different level specifications and a precise obligation for each refinement are clearly specified.

## 3 Function-Safety Interactive Development Process

Although the successive refinement model has many problems to be addressed in order to be suitable for large and complex software development, it seems to be the most solid foundation for our purpose of modelling formal software process for safety critical systems.

Based on the successive refinement model, we propose a function-safety interactive development process for developing safety critical systems, as shown in Figure 5. The user requirements are first formalised into the abstract formal specification$_1$. Then a safety analysis is conducted based on this specification. The purpose of the safety analysis is to identify more safety requirements for the system to implement and to discover whether the current specification is qualified for the safety constraints of the system. The techniques for safety analysis can vary, but we prefer *fault tree techniques* to others because the former has been used in industry for over a decade and has proved to be effective.

After safety analysis of the specification$_1$, a refinement of the specification can be carried out. During this process, both the functional requirements expressed in specification$_1$ and the safety requirements expressed by a group of fault trees must be considered as the basis of the refinement. After the refined specification is produced, the same procedure as for specification$_1$ should be performed until the final program is derived. Then test of the program against the user requirements must be conducted to dynamically confirm that the developed system satisfies the user's real requirements.

One important point about the function-safety interactive development process is that it incorporates safety analysis into the successive refinement of functional specifications of systems.

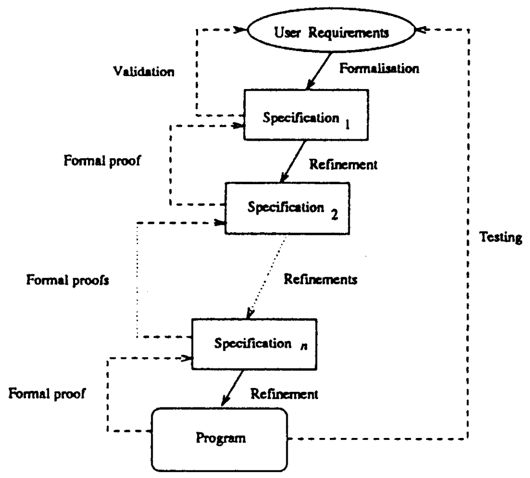

Figure 4: Successive refinement model of software development

This feature fits very well the requirements of safety critical systems. Without the interaction between safety analysis and functional analysis, the dependability of the ultimate system cannot be easily ensured.

## 4 Conclusions

We have identified several major techniques for process modelling, which include *Petri Net*, *Statecharts*, *Jackson System Development* and *Data Flow Diagrams*. Also, we have investigated several popular software process models, which are *Waterfall Model*, *Spiral Model*, *Successive Process Model* and *Successive Refinement Model*. Based on the successive refinement model, we have proposed a function-safety interactive development process for developing safety critical systems.

Our future research along this line will focus on four respects. First, develop a formal and graphical notation which should be able to be used for all the activities in every stage of formal development (e.g. analysis, specification, design, implementation). In this respect, the formalised data flow diagrams are a good candidate because (1) data flow diagrams are popular in industry for big project development, (2) we have the background of this area. Second, incorporate formal notation into fault tree notation to derive a formalised fault tree notation. Third, provide a principle of formal software process for the development of safety critical systems, which should include the principle and rules of enforcing safety constraints and formal proofs during a whole software process. Fourth, model cause-effect relationships between different activities during the formal software process for software error analysis and software reliability measurement.

## 5 Acknowledgements

We would like to thank all of our colleagues who discussed with us about the ideas presented in this paper for their constructive criticisms and suggestions. We should also thank Hiroshima City University and The Queen's University of Belfast of the U.K. for partially funding this research.

## 6 References

[Boehm 88] B.W. Boehm, "A Spiral Model of Software Development and Enhancement",

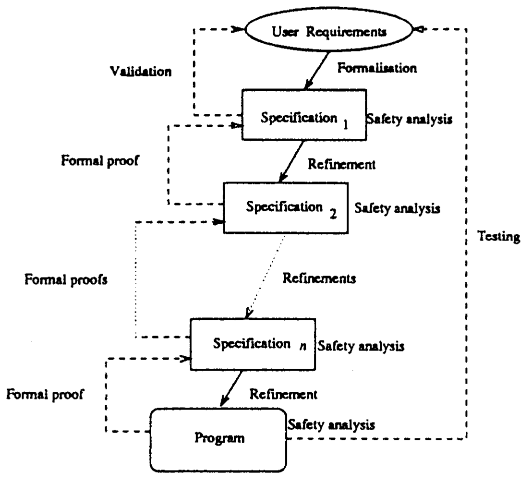

Figure 5: Function-safety interactive development process

IEEE Computer, May 1988, pp. 61-72. Reprinted in R.H. Thayer (ed) IEEE Tutorial on Software Engineering Project Management, 1988.

[Bryant 91] A. Bryant, "Structured Methodologies and Formal Notations: Developing A Framework for Synthesis and Investigation", in Z User Workshop, Oxford 1989, edited J.E. Nicholls, Springer-Verlag, Heidelberg, 1991.

[Hee *et al* 91] K.M. van Hee, L.J. Somers, M. Voorhoeve, "Executable Specifications for Distributed Information Systems", in E.D. Falkenberg, P.Lindgreen(eds.), Information System Concepts: An In-depth Analysis, North-Holland, 1991.

[Hee *et al* 91] K.M. van Hee, L.J. Somers, M. Voorhoeve, "Z and High Leve Petri nets", VDM'91, Lecture Notes in Computer Science, Springer-Verlag, 1991, pp. 204-219.

[Hoare 85] C.A.R. Hoare, "Communicating Sequential Processes", Prentice-Hall International, UK, LTD., 1985.

[Jones 1980] C.B. Jones, "Software Development", Prentice-Hall International(London) Inc., 1980.

[Jones 1986] C.B. Jones, "Systematic Software Development Using VDM", Prentice-Hall International(UK) Ltd, 1986.

[Kung 89] C.H. Kung, "Conceptual Modelling in the Context of Software Development", IEEE transaction on Software Engineering, Vol. 15, No. 10, October 1989.

[McDermid 93] John A. McDermid (ed), "Software Engineer's Reference Book", Butterworth-Heinemann, 1993.

[Milner 89] R. Milner, "A Calculus of Communicating Systems", Lecture Notes in Computer Science, No. 92, Springer-verlag, 1980.

[Royce 70] W.W. Royce, "Managing the Development of Large Software Systems", In Proceedings of IEEE WESCON, 1970, pp. 1-9. Reprinted in Thayer, R.H. (ed.) IEEE Tutorial on Software Engineering Project Management, 1988.

[Salter 93] J.E. Salter, "FASGEP Summary Report: Tasks 1-6", FASGEP Project, DTI Reference: IED4/1/9004, Lloyd's Register, U.K., August 1993.

[Whitten *et al* 93] Jeffrey L. Whitten, Lonnie D. Bentley, Thomas I.M. Ho, "Systems Analysis and Design Methods", Times Mirro/Mosby College Publishing, 1986, pp. 218-277.

[Yourdon 89] Edward Yourdon, "Modern Structured Analysis", Prentice-Hall International, 1989.