Lyee Internet Information

Requirement Engineering

Interview

With
Colette Rolland, professor at Université Paris 1
Selmin Nurcan, associate professor at the Université Paris 1
and Fumio Negoro, president of the Institute of Computer Based Software Methodology and Technology

Associate Prof. Nurcan Prof. Rolland President Negoro

Date: Friday, May 25, 2001
Place: Head office of the Institute of Computer Based Software Methodology and Technology
Moderator: Issam A. Hami, professor of the Iwate Prefectrual University

Biography of Professor Colette Rolland

1943 Born in Dieupentale, Tarn et Garonne, France

1966&1971 Awarded PhD in applied mathematics from Université de Nancy

1973 Appointed Professor at Université de Nancy, Department of Computer Science

1979 Became Professor at Université Paris 1, Department of Mathematics and Informatics

Main research area: Information System Analysis and Design, Requirements Engineering, Database Design, Case tools and CASE environments, Meta-CASE, Method Engineering and CAME environments.

Main publication:

C. Rolland, N. Prakash " From Conceptual Modelling to Requirements Engineering", Special Issue of Annals of Software Engineering on " Comparative Studies of Engineering Approaches for Software Engineering", 10, 151-176, 2000

C. Rolland, "Information Systems and Web Information Systems : A Methodological Perspective", (Invited talk), Proc. International Forum cum Conference on Information Technology and Communication at the Dawn of the New Millennium, Bangkok, Thailand, August 2000

C. Rolland, "Requirements Engineering for COTS based Systems", Information and Software Technology, 41, 985-990, 1999

Biography of Associate Professor Selmin Nurcan

1961 Born in Istanbul

1991 Awarded Ph.D in Computer Sciences from INSA (Institut National des Sciences Appliquée) of Lyon

1991 Appointed associate professor at Université Lyon 1.

1998 Became associate professor at Université Paris 1, Business Administration Institute

Main research area: Cooperative Information Systems, process modeling, CSCW

Main publications:

S.Nurcan. "Analysis and design of co-operative work processes: a framework". Information and software technology, Elsevier, 1998, 40(3), pp.143-156.

S. Nurcan. "Main concepts for cooperative work place analysis". Proceedings of the XV. IFIP World Computer Congress Telecooperation, 31 August - 4 September. 1998, Vienna, Austria, p. 21-36.

S. Nurcan, C. Rolland. "Contributions of Workflow to quality requirements". Knowledge and Process Management : the Journal of Corporate Transformation, John Wiley, 2000.

Negoro: Conventional methods usually require us to evaluate logic of the programs, but Lyee methodology is characterized by the fact that it ceases this evaluation process. In other words, the method is aimed not to rely on human thinking. It may sound a bit hyperbole, but the Lyee method is constructed to use a human capability in the least possible way for determining requirements.

Prof. Rolland: I came to Japan in order to understand the Lyee method better. Though I understand it still only a little, I feel I began to see it. The presentation we had in Tama factory this morning was very helpful because it made concrete the relationship between Mr. Negoro's philosophy and the actual work done by engineers and development sites.

Based on my current understanding of Lyee, I am tempted to paraphrase what Mr. Negoro proposes in the following way : the Lyee methodology has discovered a general structure for software to organize a program and help in its 'generation'. The program may be a set of programs or pieces of code that are incorporated into one single global structure that controls the software execution. The pieces of code are plugged into the structure. I understand that the iterated activations of this structure leads to the production of programs. Whereas the structure is generic, the pieces of code are specific to the application at hand.

Negoro: The Lyee method has thrown away an engineering aspect from itself, and it tries to capture software by using a philosophical concept. In this sense, a structure of the Scenario Function (SF) is considered the one to define requirements as a natural language does. Since the structure of the SF can be set on a computer to be executed as programs, I suppose sometimes people will take the SF as engineered programs. The Scenario Function, however, should be considered as a language structure for handling requirements. It is rather a new language cognition structure. A while ago, a concept of reengineering was vigorously discussed in the software field. In Japan as well, that idea was very popular for some time, but nobody talks about it any more. Reengineering is not regarded as a major remedy now. In other words, I want to say that conventional methods failed to find means to properly form a concept of reengineering.

Prof. Rolland: Although Mr. Negoro may disagree with it, the Scenario Function is equivalent to a framework or a pattern in current software Western terminology. It is a framework by which any software can be structured and produced. This is my understanding. A framework is repeatable and I think the Scenario Function is like that, so we can make good use of it again and again. I also think that the Scenario Function is output driven.

Prof. Hamid: It is true that the Lyee method is output-centric. That is unusually a unique characteristic. This trait is expressed in an operation of the SF and seems to work as a pattern of the framework.

Prof. Rolland: What is most interesting to me with the Lyee method is that the existence of an overall generic software structure makes possible to handle the production of a large-sized software efficiently. This is realized by adapting the generic structure to the application at hand and 'plugging' all the code items to make this structure able to generate a program. It is interesting.

We know that modularity is key to software development and in a certain sense, the Lyee method helps in the production of modular programs.

Negoro: Yesterday, Prof. Rolland presented her requirements engineering system, Crews l'Ecritoire. That was very interesting. What fascinates me most is the possibility to combine an idea, which is an extended idea of conventional methods, with the Lyee method.

The day before yesterday, we invited you to a tea ceremony. The tea house represents a concept of the entire universe. It is built on an idea that the entire universe exists in such a small space of the tea house. The reason why I mention this is that in European culture, analysis is a basis of thinking and people give some concepts to those analyzed or decomposed things. People often argue whether some systems are difficult or not, small or large, and easy or complicated. But it seems that this discussion itself does not represent an Oriental thought, but a European concept. A European way of thinking is successful in a physical world. However, I put a question mark on whether or not this European way of thinking can be applied to software filed as well, because software deals with a non-physical world. With the Lyee methodology, such a European concept is incorporated into the Scenario Function. Looking at the structure of vectors, you will see what I mean. The Lyee's structure possesses a nature of a tea ceremony

house. It is very simple and small, but it embodies the universe.

A conventional way of thinking in software is likely to conceive software as a physical entity so that it is impossible to make a conceptual comparison between a conventional and Lyee methods in the light of definition of software.

As for the largest-sized system we have developed with the Lyee method, it has the number of 1,400 screens. To complete this system with Lyee took only 10 months. The same type of system was undertaken by two other teams or corporations in Japan. The result is that they need three times larger than ours in terms of cost and period.

Prof. Hamid: I think Mr. Negoro and Prof. Rolland talk about slightly different things. First of all, we all know that it is impossible to measure how much we understand a user requirement. When we think of a system, usually a user requirement is dominant. But he/she cannot describe it well in natural language, even though he/she utilizes a model. A couple of hours' talk, even though it may result in a system in a couple of months, does not help a user fully express his/her requirement. Various relevant matters come out from the requirement. Because of that, the requirement could be complicated and developed into a large-sized system. For instance, a military system or control tower in the airport or a control system for a space station and so forth need lots of work to be done so that requirements are naturally complicated.

Shortly before, Mr. Negoro compared Lyee with the tea ceremony house from a designing point of view, but I see the Scenario Function is equivalent to an engine. The engine can operate in a sense for a compact and small requirement as well as for a large one. Putting this aside, however, I think that a requirement itself can be extremely complex and cause some problems. Of course, there are some cases in which a requirement is composed of formulated individual parts and

its combination works fine. Nonetheless, I am much interested in the work of the Scenario Function, which has a vital function, and I guess Prof. Rolland also wants to know how it works. Anyhow, I believe a certain kind of complexity exists in a requirement.

Prof. Rolland: As Mr. Negoro mentioned, it is true that when we face a problem, our French culture since Descartes has instructed us to break it down into pieces to handle it better. Thus, we are used to adopt such a decomposition approach to bring an issue into smaller ones. It is our French tradition and I do not deny it. However what interested me more is abstraction. Abstraction is not the same as decomposition or analysis. Abstraction entails for example use of generalization and classification. I spent most of my time in my professional life on abstraction. I am fully aware how effective abstraction is as a method to solve problems and I like to teach it to my students. Selmin here prefers decomposition, though.

Assis.Prof. Nulcan: Concerning abstraction, I respect an idea of abstraction, but I would rather be fit to decomposion as Prof. Rolland said. I am not very good at abstracting at the same level as decomposing. Anyway, talking about a large-sized software, I think its problems are not simply associated with decomposition, but with regeneration of once-decomposed ones. Suppose there is a business application of connecting a big company and its affiliates in terms of communications or another application of making links among organizations at the time of the M&A, we have to take global perspectives to handle the situation for our clients. Under these circumstances, I wonder how the Lyee method tackles realistic problems such as utilizing existing systems and reusing applications.

Negoro: Once we start to analyze a certain object, its related concept is expanding on and on. Then, it becomes necessary to synthesize those decomposed concepts again, because just decomposition is no good. Supposed we are dealing with some entities in which new concepts are not generated even

through the process of decomposition and composition, such an approach as decomposition and composition may work well. But just composing does not work for software. In this sense, we must create a certain type of abstraction that could work well in software. I have long been thinking of and researching into this theme.

When I was younger, I was a staff of a university, teaching software and mathematics. But the theme I wanted to study was not appropriate to a university those days. I then decided to quite the university and studied it for myself. In this sense, I do belong to neither academia nor industrial field. I have since continued to study this issue in my own way. If I had been working for the academia, I would have written the same sort of papers and stood for the same kind of views as most of the scholars do. While I was studying that subject, I was also involved in various system projects as a leader. Through these experiences, I reached a conclusion that a requirement could never be clearly determined due to its uncertain nature.

On the other hand, a notion of "difficult requirements" seems to be taken for granted, but I think this is just a comment of some people who are engaged with the "difficult requirements." Nobody can objectively judge whether they are truly complex or not. From this point of view, a concept of how to utilize engineering has yet to be elaborated on.

Prof. Hamid: I think we agree with you, Mr. Negoro, but I do not agree with you in that a requirement should not be put in a category of small or large. You seem to say that there is a category of small or large. But I do not think that a large requirement or a small requirement itself does exists. It can be small and large at the same time. This is because requirements are often intertwined at various levels. It is obviously difficult to capture a user requirement just by talking with him/her. Mr. Negoro smartly handles this problem by using a word-based structure, but requirements remain ungraspable, because they

are intertwined with each other.

Prof. Rolland tries to grasp them through natural language. Since natural language inevitably contains ambiguity, she constructed a method to remove this ambiguity, applying certain rules. The method is aimed to figure out what a user truly wants to do or their goals. Most of the goals are co-related to each other. That makes a system complicated. There are some cases in which two goals to be set are conflicting with each other so that they cannot be simultaneously attained. There are other cases in which Goal A is co-related with many other goals, with one of the goals being a parameter. If some parts of the system are changed, all the other parts of the entire system will also be affected. Such a co-relation is not so easy to be addressed in calculation formulae only. A lot of reasoning must be provided. This makes requirements big and small.

Prof. Rolland: First of all, Mr. Negoro is welcomed by the academia. We like people who base their reasoning on abstractions. Secondly, all of us dream to invent a design theory for software. But I also think that since software is not matured yet, it is too early to create design theories. In physics, it took 150 years to reach a point where a number of theories were established after its basic idea was generated. So, I suppose some theories of software may be established later in this century.

When I think in terms of software abstraction, I like to do it through software intent or intention. This is why I took interest in Mr. Negoro's idea when he and his team came to Paris to present his intention-oriented view of software production. That was exactly fitting to the way I would like to think about software. Nonetheless, I suppose since I am a woman, I have been dealing with more concrete matters than men in my daily life and profession. Thus, I am very interested to relate the intentional view to the concrete implementation view.

Having a system development project, we are often

positioned to help realize goals of the customers. This is because most of the software contributes to fulfillment of corporate goals. Thus, in the design approach developed in my group, we try to relate the software to the corporate goals and strategies to achieve these goals. In this sense our abstract view of a software is a set of organizational goals and associated strategies that the software allows to fulfil. If we want software systems to be useful, they must contribute to achieve the companies' purposes. Based on this way of thinking, we derive software requirements from corporate goals. The functionality of a software system is seen as a means to satisfying or satisficing these goals.

Lyee supposes that requirements are given. At a certain point of time, the customer and the developer have to agree with a set of requirements. Nonetheless, Lyee seems to put requirements out of the scope of its method, while we attach great importance to eliciting requirements.

Why is this problem so critical? All practice surveys show that first of all most of the failures of actual projects are caused by poor understanding of requirements. Secondly, today computer systems are more and more embedded in business processes. People want information systems which help them add value to the process. It is becoming critical to demonstrate that the system is bringing additional value to the business. This is why the link between requirements, system specification and corporate goals is essential. The third reason is that the objectives of the company are changing all the time. If you do not maintain conceptual relationships between the goals and the requirements, it becomes extremely difficult to evaluate the impact of change of corporate objectives onto requirements and subsequently on the specification of the system.

Let me add one more thing at last. Taking up a recent example, I would like to show how different what a user says he/she wants is from what the user should have in actuality. We

received a request from one software house in France who sells educational software for pupils in primary and secondary schools. That client utterly said, "we know exactly what we want" : a super operating system that can accommodate all software you can click on and use. I spent two days with them to elicit their requirements. As a result of that, it became clear that what they would like to see in actuality was not at all an operating system but a knowledge based system organized around a repository to keep track among other things, of learning activities of each pupil and their results, of relationships between pupils and teachers, among pupils, with pupils' families, and so forth. It turned out the software they need was very different from what they first wanted to have, although they spent six developing a prototype to validate their initial idea of a super operating system.

Negoro: When you are asked to consult a user requirement, the requirement will be expressed with your intention. When someone else is giving consultation on it, that requirement will be formed with that person's intention. Thus, consultation leads to a complicated story. No matter how many mistakes the first-expressed requirement may include, it is important to accept it as it is because that is close to a primary inspiration of the user. I believe that the Lyee method is able to realize the requirement in this way.

Assis. Prof. Nurcan: Nevertheless, as Prof. Rolland referred to statistics at the beginning, 50 percent of developed software is not used in reality. That is largely because users cannot express well their needs. It is a great pity to see some of the software will not be used. But this is reality.

Prof. Rolland: I understand well Mr. Negoro's approach as one of the possibilities to adapt a software to the user needs by 'code and fix'. In a RAD (Rapid Application Development) kind of development, a prototype is built and revised till it meets the user requirements. If a method provides a mechanism efficient enough to support this approach, it is fine. But this

is only possible in a rich country like Japan!!

Negoro: In my opinion, none of the software development is successful. All of them are in failure. As long as the Earth exists, failure repeats itself. The fact that produced software was abandoned in Europe is rather related to a cultural problem. I do not think that is caused by an issue whether a requirement is captured well or not.

Prof. Hamid: Before closing, I would like to hear some comments or a vision on our collaboration project.

Prof. Rolland: As for the collaboration project, I am interested in gaining a deeper understanding of how the Lyee method works. Specifically speaking, I am thinking of reengineering our tool Crews l'Ecritoire in Lyee. The tool is made with Prolog and VB currently. This attempt is good for us to be fully aware of the way Lyee works, and eventually good for you to get a software which is equivalent to Crews l'Ecritoire. You could use it for demonstration purposes or at a requirements engineering phase before the application of the Lyee method.

Negoro: Wonderful. I am pleased to hear that.

Prof. Rolland: A second idea is that through this process, we can try to evaluate Lyee's development process in our own way. We are long experienced in method modeling and meta-modeling, so we would like to apply our method engineering approach to formalize Lyee.

Through the exercise of reengineering Crews l'Ecritoire with Lyee, we can understand the process of the Lyee method and try to support it with guidelines. At the same time, I am not sure, but it may be possible to support workers in Tama factory by formalizing their knowledge and providing some heuristics and guidelines. I wish we could propose something which could even simplify the development process.

Negoro: That is a great idea. I think we can set up a different project to promote that idea.