Crossroads
The ACM Student Magazine

Planning and Improvisation in Software Processes

by ***Rosalva E. Gallardo-Valencia*** and ***Susan Elliott Sim***

Abstract

This paper presents the results of an empirical study aimed at examining the extent to which software engineers follow a software process and the extent to which they improvise during the process. Our subjects tended to classify processes into two groups. In the first group are the processes that are formal, strict, and well-documented. In the second group are the processes that are informal and not well-structured. The classification has similar characteristics to the model proposed by Truex, Baskerville, and Travis [**12**]. Our first group is similar to their methodical classification, and our second group is similar to their amethodical classification. Interestingly, software engineers using a process in the second group stated that they were not using a process. We believe that software engineers who think that they are not using a process, because they have the prevalent concept of process as something methodical that is strict and structured, actually are using an informal (amethodical) process. We also found that software engineers improvise while using both types of processes in order to overcome shortcomings in the planned path which arose due to unexpected situations. This finding leads us to conclude that amethodical processes are processes too.

Introduction

When going on a trip, it is often necessary to plan ahead. How will you get there? Nowadays, it is common to get directions from online map Web sites, such as MapQuest [**8**] or GoogleMaps [**6**]. These systems give you detailed written directions, a visual map, and also the estimated duration of your trip. At the bottom of the page, there is a disclaimer:

"These directions are for planning purposes only. You may find that construction projects, traffic, or other events may cause road conditions to differ from the map results" [**6**].

In normal or ideal conditions, you can follow the directions, but conditions are not always normal. What would happen if one of the suggested roads is closed for maintenance? In this situation, you cannot follow the directions and have to take an alternate route to get to your destination. Software engineers have to face similar problems when following a software process. Software processes provide guidelines that engineers can follow in general. However, during development, unexpected conditions might occur, such as changing requirements or insufficient personnel. Due to these conditions, software engineers have to take a detour from the planned path, just like in our road trip scenario.

We conducted an empirical study of software engineers in order to investigate the extent to which they follow a planned software process path. We found that software engineers frequently claimed that they did not follow a process, especially when they used an informal process, but, in reality, they did. At the same time, software engineers frequently had to improvise processes, even when they were following informal processes. While these results make sense intuitively, i.e., that people both plan and improvise, they contradict prevailing views of software processes. Using the analytical lens provided by Truex, Baskerville, and Travis regarding the nature of software development [**12**], we conclude that processes and improvisation are both methodical and amethodical. It is important to clarify that we refer to an informal or amethodical process as a process that is not strict, well-structured, or well-documented. This definition does not imply that practitioners of this type of process are less careful, in comparison to practitioners who use a formal process.

Truex et al. provide a model of development methods based on deconstructing the meanings within different descriptions of process. They arrive at this model by analyzing descriptions of methods and discerning the central and marginal meanings in these descriptions. Based on this analysis, they argue that the central message of a process description is methodical, but the marginal message is amethodical. The central message is found in the explicit content of the description, but the marginal message is what is missing or excluded from the description. A development process is methodical in the sense that it is logical, predictable, and controllable; this is the central message. In contrast, the marginal message is that a development process is capricious, fragmented, creative, ad hoc, and social.

Software engineers commonly associate the concept of process with a formal or methodical process. The word "process" connotes a procedure that is planned, documented, structured, and strict. Consequently, processes that do not have these characteristics have been marginalized, which Truex et al. label as being amethodical. Our study provided evidence consistent with this dichotomy. We found that software engineers who were not using a methodical process stated that they were not using a process at all. We also looked for evidence of improvisation in software processes, and we used the presence of improvisation as an indicator that a process was being used. In other words, improvisation only occurs when actual circumstances deviate from the plans; therefore, its presence shows that there was a planned process that was either methodical or amethodical. We found that improvisation occurs in methodical process, as was expected. However, we also found that improvisation was also needed when people developed software "without" a process. Consequently, we concluded that the marginalized concept of

process, an amethodical process, is also a process. In our case, software engineers claiming that they were not using a process were really using a process and improving it at the same time.

In the next section, we explain the characteristics of methodical and amethodical software development. The Methods section describes in detail the research methods used during this study. The Results and Discussion section presents our findings, and the Implications section presents the usages and lessons learned from our results. Finally, we conclude our study in the Conclusions section.

Background

This study is concerned with software process, which Fuggetta defines as, "the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product" [**5**]. This is a typical definition of software process that emphasizes the predictable and regular activities in software development. Moreover, the emphasis on these aspects of software process often leads people to believe that a true software process is esoteric and unobtainable. Formal descriptions of software process models tend to be abstract and general. As a result, there are gaps in what happens when these processes are enacted by people in specific circumstances.

Consider the following example. Lois, a subject in our study, was having problems with a high defect rate in the source code. She works in the aerospace industry, where the software process tends to be highly structured, and software quality is important. However, the prescribed process was not adequate to the task. Her solution was to add an informal inspection activity, in which one programmer explained his task to other programmers, who peer-reviewed the code. This activity improved the quality of the software by finding bugs early. This situation shows that Lois usually follows a routine to develop software, but there are situations where the prescribed steps fell short. In these cases, Lois needed to use her creativity to improvise on the software process to find a solution. This is precisely where there is a gap between process models and enactments of these processes.

Dyba gave the following description of improvisation in software processes: "Improvisation deals with the unforeseen. It involves continual experimentation with new possibilities to create innovative and improved solutions outside current plans and routines. The explorative nature of improvisation necessarily involves a potential for failure, leading to the popular misconception that improvisation is only of a spontaneous, intuitive nature that happens among untutored geniuses or in immature organizations. However, organizational improvisation does not emerge from thin air. Instead, it involves and partly depends on the exploitation of prior routines and knowledge" [**4**].

Others have observed the interplay between prescription and improvisation. For instance, Brown and Duguid [**3**] stated that it is necessary to have both prescription and improvisation in mind while managing organizations. They conducted a study on

customer service representatives who fix Xerox machines. They observed that workers often could not follow the prescribed process because some characteristics of the machines were not considered in the manual, such as the age of the machines. In those situations, workers used the knowledge of other coworkers and improvised a solution. The authors concluded that having a balance between prescription and improvisation helped organizations to be creative and innovative.

Suchman [**11**] explained the tension between plans and situated actions in artificial intelligence research. She defined plans as prescribed actions, in other words, the set of appropriate actions for typical situations. On the other hand, she introduced the term "situated actions" to describe the fact that each action depends on social factors and resources that a particular occasion provides. We can relate the concept of plans with prescription and the concept of situated actions with improvisation.

Truex et al. characterized prescription and improvisation as methodical and amethodical software development. They used as their starting point a historical definition of method as the steps that are taken to accomplish a particular task. These sequences of steps, and, in turn, the definition, became more abstract and general, so that a method was a prescription for a solution. The concept of amethodical software development came from applying a deconstructionist literary analysis to writings on software process. This technique proceeded by identifying a central, dominant, or privileged message in the text and, subsequently, inferring a marginal or excluded message. Using this technique, they arrived at the term "amethodical" software development to denote the ways in which software was developed, but had been excluded from the privileged text, i.e., the formal descriptions.

Methodical software development is characterized as controllable, sequential, universal, and goal-oriented. This type of software development is controllable because it is possible to manage the pieces of software or process obtained through decomposition. It is sequential because it consists of ordered stages that have to be followed in a linear manner. It is universal because it could be applied to different organizational settings, and it could be equally successful. It is goal-oriented because this type of process is based on the assumption that some goals have to be defined, and they should be achieved after applying the process. One example of this type of software development is the waterfall process [**9**]. This software development process can be considered methodical because it proposes to follow the stages of analysis of requirements, design, implementation, testing, integration, and maintenance in a sequential and universal manner.

On the other hand, the main characteristics of amethodical software development are that this type of software development is opportunistic, fragmented, ad hoc, and social. It is opportunistic because it appears as a consequence of the current and unanticipated situations. It is fragmented because it cannot follow a linear pattern. The stages appear independently and can occur in parallel. It is ad hoc because this type of process depends on the local settings of the organization. Instead of following plans, it follows situated actions. It is social because people are an important factor in this kind of process. Some examples of this type of software development include the agile software processes such as extreme programming [**2**] and Scrum [**10**]. The agile software process, as stated in the Agile Manifesto [**1**], share many characteristics of amethodical software development.

This type of process is opportunistic and ad hoc in that it can adapt to changing circumstances. It is highly social in that it relies on knowledge obtained from communications between customers and developers, rather than written standard documentation.

A table from Truex et al. has been reproduced below in order to summarize the differences between methodical and amethodical software development [**12**].

Table 1: Assumptions and ideals of methodical and amethodical processes.

Methodical	Amethodical
1. Information systems development is a managed, controlled process Idealizing: logical decomposition and reductionism	2. Information systems development is random, opportunistic process driven by accident Idealizing: holism and creativity
3. Information systems development is a linear, sequential process Idealizing: temporal causal chain	4. Information systems development processes are simultaneous, overlapping and there are gaps Idealizing: fragmentation, parallelism, and disconnectedness
5. Information systems development is a replicable, universal process Idealizing: generalization, consistency, and formalisms	6. Information systems development occurs in completely unique and idiographic forms Idealizing: choice, change, and adhocracy
7. Information systems development is a rational, determined, and goal-driven process Idealizing: goal predetermination, process	8. Information systems development is negotiated, compromised, and capricious Idealizing: conflict, social constructivism, and human independence

predetermination, and human cooperation	

Having laid out these concepts and having explained what we mean by improvisation applied to software process, we will discuss our methods and findings.

Methods

Subjects

The target population for this study was software engineers who are currently following development processes while developing or managing significant software projects. A total of eight software engineers participated in this study.

The participants were between 25 and 41 years old. Six of them hold a BS degree in computer science, one holds a BS in aerospace, and another holds an MS in computer science. Their experience in developing software ranged from 4 to 15 years. Six are men and two are women. Refer to Table 2 for more information about our subjects, identified with pseudonyms.

Table 2: Subjects in the study.

Subject	Years	Years Working	Process Used	Current Position	Type of Projects
David	37	15 years	Group of individual process	Software Developer	Development of software products
Joao	25	5-6 years	No process/Extreme programming	Research Assistant in University	Financial projects
Lois	28	5 years	Structured process	Software Architect	Gathering requirements to create electronics in an aerospace company
Nando	30	6 years	Structured process	Project Manager	Software for telecommunication companies and banks
Dominic	30	6 years	Process for each project/Structured methodology	Software Engineer. Position depends on project	Consulting companies
Yasin	41	8 years	Rapid prototyping/Structured process	Research Assistant in University	Research Programmer and Computer Support
George	31	4 years	Trying to apply agile methodologies / Rational Unified process (CMM5)	Technical Project Leader and Senior Developer	Development of J2EE services
Sylvia	29	5 years	Code fix/Waterfall	Research Assistant and Graduate Student	Developing a tool for programmers

Data Collection Procedure

We conducted interviews with each of the eight software engineers over a period of three weeks. The semi-structured interviews lasted between 20 and 44 minutes and consisted of six questions:

1. Could you tell me about your job?
2. Could you tell me about the procedure that you use to develop software?
3. Have you ever had a situation where the procedure didn't fit or needed to be changed?
4. What was the problem? How did you overcome it? Did you invent a new procedure?
5. Do you think that these problems had an impact on the success of the project?
6. Do you think that these kinds of problems are common in your work?

Follow-up questions were asked when appropriate.

Although the study was about improvisation while using a software development process, the word "improvisation" was not used in the questions because this word can have a bad connotation. However, in some cases, our subjects voluntarily mentioned this word. In that case, we continued using it during the interview.

Data Analysis Techniques

During the transcription of each interview, the text was analyzed using "focus coding" [7]. Focus coding is a technique that allows pieces of information in the transcript to be coded in order to highlight points of interest for our study. For example, something that we were interested in was the feelings that our subjects expressed during the interview. Focus coding allowed us to probe this topic in depth and to examine our subjects' feelings regarding deviation from a software process. An example of focus coding is shown in Table 3. In this table, the first column is the transcription of the interview, and the second column represents the analytical theme to which some lines of transcription are related.

Following transcription and coding, we cross-tabulated the codes between files in order to categorize our findings. For example, we looked in other transcripts for pieces of information that were related to the expression of feelings by our subjects.

Table 3: Example of transcription and focus coding analysis.

04:20 **So the analysts that gather the requirements they produce a**	

document? Some did, some did not (laughing) **They produce something written and they gave you the document ?** They will meet for this and talk about that with the developers.	Nervousness for not following the procedure
04:42 **And after that the developers have like a standard to code or some guidelines?** Coding standard? Some places we did not even have that but some places do. **Ok and after that how do you report the hours that you put in your work.** I was usually in at an office so, we knew more or less how many hours we spend in each feature or whatever. That is. In some places I was working from home and I would report the hours count them not very accurate but in a reasonable range of how many hour I spend in each task and then I will report them.	Coding standards: not always Reporting hours from home and office
05:44 **After that, when you finish the code you started to test. How will you know if something is done** Well (long pause) when it was ... Just like people asked me or what I have understood what people asked me I will deliver it. Maybe it was not exactly what I did, maybe I understood something wrongly and then it will get back to me and do what people ask me or maybe the client did not like or wanted to change the software and then.	Developer understands the requirements

Threads to Validity

Although this empirical study was conducted with eight subjects, having differing backgrounds, we are aware that this study has some limitations. First, the authors of this paper contacted the subjects, conducted the interviews, transcribed the interviews, coded the transcriptions, and analyzed the information. It is possible that this fact has induced a bias in the analysis. In addition, it could be possible that different people could code the transcripts in a different way or even that other researchers could analyze the results with a different perspective.

Second, the size of our sample is small. Different results could have been obtained with a larger sample. Another issue is the years of experience of our subjects. They have, on average, 5.5 years of experience, except for one who has 15 years of experience. We suspected that having subjects with more experience could change our findings. Finally, it is important to note that not all of our subjects are working in industry. Some are working in academia, which could have affected our results.

Although our study has some limitations, the results obtained represent the findings of an initial study and provide us with some useful empirical data to evaluate our conceptions about the nature of software processes.

Results and Discussion

In this section, we present the findings of our empirical study about the extent to which software engineers follow a software process and improvise while using a process. We found that all of our subjects used some kind of software process. These processes were either formal or informal. These two categories align closely with the methodical and amethodical aspects of software development. In addition, subjects who used an amethodical process believed that they were not using a process. Moreover, we found that no matter which type of process the software engineers used, they needed to improvise. They did so to deal with shortcomings of the process in daily activities and during unexpected situations. The presence of improvisation in both types of processes helped us to determine that amethodical processes are, in fact, processes too.

Processes are Both Methodical and Amethodical

We asked our subjects about the software process that they are currently using. Almost all of them used a classification of software processes that fell into two general groups. The different classifications mentioned by our subjects are shown in Table 4.

Table 4: Software process classification.

Type A	Type B
Formal	Informal
Formalized and well-documented	Chaotic
Strict	Non-strict
Structured	Non-structured

Yasin clearly stated during the interview that he divided software development processes into two main categories. He said, "Well, yeah. It is a process depending upon people instead of a formalized, well-documented process. It is a chaotic process." From his words we can distinguish two opposite types of processes: "formalized, well-documented" and "chaotic." He had a clear separation of concepts between the processes that formally impose the creation of documentation and the processes that do not. He referred to the second group as chaotic. Interestingly, the process that he used fell into the chaotic group.

When Sylvia talked about the software process that she was using, she mentioned the characteristics that her process does not have. In this way, we can imply the characteristics of the other group to which her process belongs. Sylvia said, "Yeah, it could be a process. It does not have a name, but... it is not a strict software engineering process in terms of based on phases. It is not very defined." She emphasized that her process is "not a strict" process and that it is "not very defined." These counter-characteristics helped us to imply the characteristics of the other group of software process—strict and well-defined.

We observed that our subjects classified software processes into two general groups. They used different terms to label this classification, but the idea behind each type was the same. In Type A, we had the processes that were formal, documented, and followed in a strictly structured manner. In Type B, we had processes that were exactly the opposite. These processes did not enforce strict documentation or structure to be followed. Although these processes are classified into two different groups, they both define steps to be followed while developing software. Therefore, both can be considered software processes.

In our study, Lois and Nando used a mandatory structured (Type A) process. In addition, Dominic used a process that was defined by each project in its proposal (Type A). Our other five subjects claimed that they do not use a process to develop software. For them, using a process that does not enforce documentation or a strict structure means not using a software process at all. After asking some questions about the steps to develop software, we could see that they do have a process, but it is not a Type A process. Ultimately, they recognized that they have some kind of process.

George initially said that he was not using a process. When we asked him whether he was sure that he was not using a process, he said, "Yeah, well, it is not a formal process, but it is a process with which we feel comfortable." He did not sound convinced about that when he said, "Yeah," and he continued, "it is not a formal process." For him, it was difficult to recognize that he used a process.

David, in addition, expressed his uncertainty about the effectiveness of using a Type B procedure. He said, "It is sort of an in-house process developed over time. Nobody will say this is the process and we should follow it. It is like multiple software development processes that are individual.... Sometimes I am amazed that we can deliver a product with this process." Although his company delivers quality software, he was surprised that this process could have successful results. He admitted that he used a process in spite of the fact that it was not a Type A process.

Joao showed some nervousness by laughing while replying that he did not use a Type A process. He said, "Well (laughing) I do not know if I do use a formal process. I mean I know that I do not use a formal software method. I just think about what I have to do and then (laughing) do it. Then test and work until it is done." Joao believed that he was not using a process because he knew that the steps he was following were not part of a formal process. Instead, they were what he needed to do in order to deliver the product.

Truex et al. proposed a model in which they classify software development into two groups, methodical and amethodical. We observed that the classification of software processes found in our study is similar to the model proposed by Truex et al. The Type A processes in our study are structured and documented. These characteristics give us the notion of a controllable process, which is one of the main characteristics of the methodical group.

In a similar way, our Type B corresponds with the amethodical group. Processes in Type B are ad hoc, as mentioned by two of our subjects. David explained that he used software processes that were suitable for each individual in his team. Sylvia also mentioned that the process she used was not very defined, and it could change depending on the circumstances. Yasin mentioned another characteristic of this group of processes. He claimed that the process he used was based on people. This characteristic reminded us of the social aspect of the amethodical processes. Because of the similarity of our findings with Truex et al., we will refer to Type A as the methodical group and Type B as the amethodical group in the rest of the paper.

Improvisation Occurs in both Methodical and Amethodical Processes

Our subjects developed software or managed software projects on a daily basis. We have seen in the previous subsection that they all use a software process, whether formal and structured or not necessarily formal or documented. In this subsection, we argue that, in all cases, software engineers felt the need to improvise in order to overcome unanticipated problems, because

neither methodical nor amethodical processes provide sufficient guidance in all situations. This fact further supports our claim that the informal processes used by our subjects should also be considered as processes. We illustrate our findings using quotations from four subjects. Two used methodical processes and another two used amethodical processes. This pattern was found in all of our subjects regardless of the processes used.

We begin with two subjects, Lois and Nando, who used processes in the methodical group to show their need to change the process when they face a problem. They had to find creative solutions when confronted by unexpected situations.

Recall the example provided about Lois in the Background section. She stated, "If I found a problem, I can take the initiative of fixing the process and then it will go to the approval board, and you have to talk with people that will be affected by the change. If the problem cannot wait, I try to solve the problem with other people, talking with other people, asking." Lois explained that she faced a problem, and she had two options to solve it. If the solution cannot wait, she would talk to people to find a solution. If the solution can wait, she will go to the normal process of approval. During the interview, she mentioned how she first improvised to solve a problem, and then the solution became part of the process. Her solution had good results, and, after a while, it was included in the formal process. For her, improvising was a way to improve the process and make a contribution for her company.

Nando commented that the process that he uses is the same for both big and small projects. He explained, "We are changing things in order to fit the small projects in our methodology, because now we spend more time documenting than changing the code for these small projects." He was having problems with small projects because he had to create a lot of documentation. In many cases, documentation took more time than the development itself. Now, when he has to implement a small project, he does not follow the guidelines of the process, but does all of the documentation at the end of the project. He had to improvise this solution because his customer asked for a quick implementation, and the customer did not care about the procedure. Nando is working with his manager to include a special procedure for small projects.

This result is consistent with partitioning provided by Truex et al. It is not surprising to find that software engineers using methodical processes, which impose a sequence and structure, have the need to improvise. They have a written process to follow, but not all of the situations that they could face are written there. They have to be creative and take a detour from the prescribed process with the aim of solving unexpected circumstances in which the planned guidelines of the process do not help.

More surprising, subjects who used a process in the amethodical group also improvised to improve their process. In particular, George and David gave clear examples. George mentioned that when his team defined the interfaces in the design phase, they preferred to do the design of interfaces iteratively and improve the design in each meeting, instead of having strict guidelines to follow. He said, "Yes, we improvise because the way we came out with the interfaces is a gradual process of improvising the

interfaces with brainstorming creating new things. It is always an increment from meeting to meeting and step by step. It is not like you freeze something and you are done. It is not like that." Although this situation could be seen as a design improvisation within the process, it is a good example of how improvisation is needed in amethodical processes where iterative design is used. George emphasized "creating new things." For him, letting developers express their creativity was an important factor, which is evidenced when they improvise while designing interfaces.

David expressed a similar sentiment when he stated, "The process that we have is so flexible that you are changing all the time. It is sort of a ... maybe is not a process in any way, so it is sort of a how the things go and come up." He described the process that he used as very changeable due to the "flexible" nature of it. It is interesting that he mentioned that "maybe [it] is not a process in any way" because it reveals that, for him, a process is something rigid that could not let him improvise. In this case, the process that he used let him do it, so he doubted if the process that he used was really a process.

We expected that software engineers would have the need to improvise when they were using a prescribed process. Instead, we found that they used both methodical and amethodical processes and improvised in both. The fact that software engineers improvised while using an amethodical process, which is ad hoc, flexible, and not structured, provides further support to our claim that amethodical processes are processes too. Our subjects faced situations where their plans were not suitable, and they felt the need to improvise to overcome the shortcomings of their process.

Implications

The results of this study can be used in different ways. First, the fact that software engineers who use an amethodical process state that they do not use a process, helps us to understand the common connotation of the word "process" among software practitioners. Software engineers are more used to the prevalent description of this word as related to methodical processes. Due to the marginalized position of amethodical processes, they have not received enough attention from researchers. Thus, this fact could be considered as a point of failure in process research.

Second, our results show that improvisation is needed in not only methodical processes, but also amethodical processes. This finding could be used for agile researchers or investigators of other amethodical processes with the goal to provide amethodical software practitioners tools and process models that support improvisation and provide the resources to innovate, while still using a software process.

Conclusions

Our study shows evidence consistent with the model proposed by Truex et al. that software engineers classify software processes into two main groups with similar characteristics. All of our subjects use some kind of software process—either ones

that are formal, strict, and well-documented (methodical) or ones that are informal and not well-structured (amethodical). It is interesting to see that the opportunistic and social nature of amethodical processes makes software engineers think that they are not using a process when they really are following one. This is because software engineers relate the term software process to the deep-rooted definition of this term, as a set of prescribed guidelines. This is the prevailing concept of software process; it makes practitioners believe that the informal process is not a real software process. In addition, our subjects seemed to relate formality with effectiveness. Although they were working with an amethodical process and having good results, they perceived that methodical processes were more effective. David, who is using an amethodical process, gave us a clear example of this situation. He was amazed by the fact that his company can deliver a product with the process they use.

We also concluded that improvisation occurs both in methodical and amethodical software processes. This fact supports our conclusion that informal processes used by our subjects are processes too. We expected to find improvisation in methodical software processes, where software engineers have to follow prescribed plans that often do not describe the guidelines to be followed in unexpected situations. Surprisingly, we found that software engineers following an amethodical software development process also needed to improvise. Although the process they applied was flexible and ad hoc in nature, software engineers still needed to innovate in order to face unexpected situations. We observed that a software process should have a balance between planned activities and improvisation. Thus, methodical processes, characterized by guidelines, need a balance with unplanned actions or improvisation. Amethodical processes, characterized by improvisation, also need some planned activities to be used as a starting point for the improvisation.

The interesting results obtained in this initial study have motivated us to continue our research. We plan to interview people using traditional software development processes (waterfall or unified process) and agile software development processes (extreme programming or Scrum), with the goal of making a comparison on how these two groups of software engineers improvise while developing software.

References

1. Agile Alliance, "Manifesto for Agile Software Development," <http://www.agilemanifesto.org/principles.html> (5 April 2007).
2. Beck, K. *Extreme Programming Explained: embrace change.* Addison-Wesley, MA, 2000.
3. Brown, J.S. Balancing act: how to capture knowledge without killing it. *Harvard Business Review,* (May-June 2000), 3-7.
4. Dyba, T. Improvisation in Small Software Organizations. *IEEE Software,* 17(5), (September/October 2000), 82-87.
5.

Fuggetta, A. Software process: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM Press, New York, NY, 25-34.

6

Google Inc, "Google Maps," <http://maps.google.com/> (5 April 2007).

7

Lofland, J., Snow, D., Anderson, L., and Lofland, L. "Chapter 8: Arousing Interest." In *Analyzing Social Settings: A Guide to Qualitative Observation and Analysis.* Wadsworth/Thomson Learning, Belmont, CA, 2006.

8

MapQuest Inc, "MapQuest," <http://www.mapquest.com/> (5 April 2007).

9

Royce W.W. Managing the Development of Large Software Systems: Concepts and Techniques. In *Proceedings of IEEE WESCON*, CA, 1970.

10

Schwaber, K., and Beedle, M. *Agile Software Development with Scrum.* Prentice Hall, New Jersey, 2001.

11

Suchman, L. *Plans and Situated Actions: The Problem of Human Machine Communication.* Cambridge University Press, UK, 1987.

12

Truex, D., Baskerville, R., and Travis, J. Amethodical Systems Development: The Deferred Meaning of Systems Development Methods. *Accounting, Management and Information Technology,* 10(1), (2000), 53-79.

Biography

Rosalva E. Gallardo-Valencia (**rgallard@uci.edu**) is a PhD student in the Donald Bren School of Information and Computer Sciences at the University of California, Irvine. Her research interests include program comprehension and software processes. Web site: **http://www.ics.uci.edu/~rgallard/**.

Susan Elliott Sim (**sesim@uci.edu**) is an assistant professor in the Donald Bren School of Information and Computer Sciences at the University of California, Irvine. Her research interests include program comprehension, empirical studies, and software processes for small businesses. Web site: **http://www.ics.uci.edu/~ses/**.