SYSTEM ANALYSIS AND DESIGN IN A LARGE-SCALE SOFTWARE PROJECT: THE CASE OF TRANSITION TO AGILE DEVELOPMENT

Yael Dubinsky
Department of Computer Science (adjunct lecturer), Technion – Israel Institute of Technology
yael@cs.technion.ac.il

Orit Hazzan
Department of Education in Technology & Science, Technion – Israel Institute of Technology
oritha@tx.technion.ac.il

David Talby
MAMDAS – Software Development Unit, Air Force, IDF, Israel
davidt@cs.huji.ac.il

Arie Keren
MAMDAS – Software Development Unit, Air Force, IDF, Israel
ariekk@netvision.net.il

Keywords: System analysis and design, Agile development, Project specifications, The system analyst role.

Abstract: Agile software development methods mainly aim at increasing software quality by fostering customer collaboration and performing exhaustive testing. The introduction of Extreme Programming (XP) – the most common agile software development method – into an organization is accompanied with conceptual and organizational changes. These changes range from daily-life changes (e.g., sitting together and maintaining an informative project environment) and continue with changes on the management level (e.g., meeting and listening to the customer during the whole process and the concept of the whole team which means that all role holders are part of the team). This paper examines the process of transition to an agile development process in a large-scale software project in the Israeli Air Force as it is perceived from the system analysis and design perspective. Specifically, the project specifications of the agile team are compared with those of a team who continues working according to the previous heavyweight method during the first half year of transition. Size and complexity measures are used as the basis of the comparison. In addition to the inspection of the specifications, the change in the role of the system analysts, as the system analysts conceive of it, is examined.

1 INTRODUCTION

System analysis and design are basic activities in software development. Traditionally, they are carried out by a separate group of practitioners, who gather the system requirements, analyse them and prepare the specifications documents to be handed to the development group. In large-scale software projects these activities are highly significant.

The agile development methods (Highsmith, 2002) and specifically Extreme Programming (XP) (Beck, 2000, 2005) introduce a change in the software development environment. For example, working according to primary practices of XP, the team *Sits Together* while implementing the notion of *Whole Team*, which means that role holders, such as the system analyst, work with the other role holders – developers, testers and team leaders. These work habits are introduced to foster communication among teammates.

A natural question to be asked at this point is: How can we deal with these notions in a large-scale project? Should system analysts need to *Sit Together* with the development team? What is the *Whole Team* with respect to system analysis and design and how this notion is interpreted in a large-scale software project?

Another example is the XP primary practice of *Weekly Cycle*. According to this practice, the work is planed on a weekly basis in accordance with full customer collaboration. In this case, we should ask: What is the role of the system analysts in these weekly planning sessions? Are system analysts the mediators or do they listen to the customer together with the *Whole Team* (i.e., developers, testers, and so on)? How do we expect the project specifications to be expressed in an agile environment?

This paper presents a field research conducted in a large-scale software project in the Israeli Air Force. The research examined the process of transition from heavyweight software development to agile development. Focusing on the system analysis and design aspect, the research aims at answering questions such as above-mentioned ones.

In Section 2 we elaborate on the transition process and in Section 3 we explain the research setting for its investigation. In Section 4 we present data analysis by comparing the agile project specifications with those of a team which continues working according to the previous heavyweight method. The comparison relates to the first half year of transition. Size and complexity measures are used for the comparison. In addition, in Section 4 we present data and analysis with respect to the change in the role of the system analysts as they conceive of it. In Section 5 we conclude.

2 THE TRANSITION PROCESS

The in-transition software project that this paper focuses on has been developed by about eighty skilled system engineers, system analysts, developers and testers, organized in a hierarchical structure of small teams. The project develops large-scale, enterprise-critical software, intended to be used by a large and varied user population.

The army is known as a large and hard-to-change organization with respect to fixed regulations, project approval, management methods and organizational structure and culture. However, when the project leadership decided to change the software development method in order to cope successfully with the challenges that the project set, the Air Force leadership supported the decided-upon transition as a mean to improve software process and quality.

After several months during which the fitness of different development methods to the said project had been investigated, XP was selected to be implemented and a pilot team of fifteen people was established and started working according to the agile method. All the other teams of the project continued working according to the previous heavyweight method.

It is important to note that during the years prior to the transition, tools and procedures were developed and used by the people in this software unit. Though it was accepted that agile development can improve the process, it was also agreed that there are tools and procedures that will not be changed at the current stage, whether because they are good practices or whether because of time constraints.

The software project is built based on a large-scale in-house object-oriented framework (Mohamed, Schmidt and Johnson, 1999), which handles many of the underlying technical aspects of the system. One aspect is the *formal detailed specifications*. This framework relies on a metadata repository (Talby et al, 2002), which contains most of the system's specifications: data entities, data types, actions, transactions, user types and privileges, messages, external interfaces and so forth. This data is edited in the repository, in formal forms – in contrast to free-text documents – and much of it is used to automatically generate code and other files.

As a result of working with this framework, the process of development starts with design, continues with writing the formal detailed specifications in the metadata repository, and then coding those parts of the specifications that are not automatically generated. In such a process, the specification writers have to be formal and precise, and as formality increases, the cost of communication increases when teams later on communicate in order to clarify loose ends.

During the transition process all teams in the project, including the agile team, continue working with formal detailed specifications and with the respective tools that support them.

The roles involved with system analysis and design in this project are architects, operational system analysts, functional system analysts, and system engineers. In this work we focus on the operational and functional system analysts. The operational system analysts are practitioners in the operational aspects of the project subject matter and are part of an operational analysis group. They define the system to be developed and they represent the customers and users. The functional system analysts process the operational specifications and convert them into engineered technical specifications. They are part of the development group.

The change for role holders stems from the change in process. As part of the transition process, only operational and functional meta-specifications are produced, and then delivered to the agile team who together with the customer and system analysts produced the detailed specifications for both operational and functional aspects. The impact of this process on system analysts is elaborated in Section 4.

Before we delve into the details of the research findings, we present the difference approaches reflected by the two development approaches as they were described by one of the system analysts who was involved with the transition process in general and with the agile team in particular. Using the metaphor of trips he says that the heavyweight software method is like an organized tourist trip while the agile method is more a journey-like trip. Specifically, the tourist makes decisions long time before executing them, plans ahead into the small details, and has not much tolerance for changes; The journeyer, in contrast, is flexible and open to changes, makes decisions closer to their carrying out, knows in general terms what he/she wants to see and plans the details only during the journey itself.

3 RESEARCH FRAMEWORK

The exploration of this transition process started two years ago when it was decided to change the traditional heavyweight software method that had been used in this organization for many years. In previous work, we presented the way agile and XP were introduced into this project (Dubinsky, Hazzan and Keren, 2005) together with the set of product and process metrics evolved in the first release of the pilot team and that guided in practice the development method (Dubinsky, Talby, Hazzan and Keren, 2005).

Within this research, the sub-research that focused on the expression of the system analysis and design aspects in the transition process, two research approaches were used.

The first approach is a quantitative comparative one, by which we aimed at measuring the implications of the transition to the agile method on system analysis and design. Accordingly, we examined and compared the specifications produced from both kinds of teams – the traditional one and the agile one. Thus, one of the main contributions of this research is the comparative data and field-based evidence it provides with respect to the role and functionality of system analysis and design in an agile XP large-scale project in a large organization.

The second research approach was a qualitative approach in which we seek to understand the process from the system analysts and designers' point of view. Accordingly, we interviewed system analysts and asked them questions such as "Do you feel that your role has been changed? If no, please describe your role before and after the transition. If yes, please describe how your role has been changed.", "Please compare the traditional way with the agile XP one.", and so forth.

In what follows the research tools are presented. For the comparison purposes, we look at two different sets of specifications. The first set belongs to the team which worked according to the heavyweight method and during the examined half year was in the phase of fault corrections before delivery. The second set belongs to the team which worked according to the agile method and during the examined half year developed three release – the second, third and fourth releases – which were each two months long, and composed of 4 two-week iterations.

It is clear that a comparison of the specifications of two different products of two different teams is not a trivial matter. Therefore, we searched for trends and relative-to-size measures rather than absolute numbers. In addition, the comparison value increases because the two teams work in the same organization, according to the same procedures, with the same infrastructure and tools, and with people with similar experience and expertise.

Three measures were taken from each set. The first measure is the size of the specifications which is used for comparison alignment. The second and third ones are two measures which are used to assess the complexity of the specifications; one of them is inspired by the measure of code cyclomatic complexity (McCabe, 1976; Watson and McCabe, 1996). In Section 4.1 these measures are elaborated and illustrated.

In order to learn about the transition process from the point of view of the system analysts, three thorough interviews were conducted. The interview was composed of five parts as follows. The first part was an introduction in which we explained the goals of the research and the interview, ask permission for videotaping, and answer questions if exist about our research and about the interview. The second part concerns with the interviewee's current position in which we ask to describe the current role and the significant and interesting things as well as the problems that occurred as part of this role before and after the transition. The third part addressed the system analyst role in general in which the interviewee was asked to define the role, draw the position of this role in the organization, and reflect about the drawing. The forth part of the interview focused on the agile environment. The interviewee was asked to share with us his/her knowledge about the agile method in general and its agile implementations in the organization in particular. Then the role of the system analyst was discussed with respect to the agile environment and with respect to the drawing from the previous part. In the last part of the interview, the interviewee was asked to imagine that she or he are going to establish a new software company and to decide about the desired skills and education of the system analysts who they will hire.

The qualitative data that is the outcome of these interviews was interpreted by using the theory of coping with change (Plotkin, 1997) and by using a reflection ladder (Schön, 1983, 1987; Hazzan, 2002; Tomayko and Hazzan, 2004).

4 DATA ANALYSIS

The data presented and analyzed in this section was gathered as described in Section 3. In Section 4.1 the project specifications are compared using size and complexity measures. In Section 4.2 the change in the role of the system analysts is examined. We note that this research still continues in order to deepen our understanding of the transition process from additional perspectives.

4.1 Specifications Comparison

The examined specifications are divided into modules. We denote the specifications of the team which worked according to the heavyweight method by *SpecH* and the specifications of the team which worked according to the agile method by *SpecA*. During the half year we took three main measurements – in the beginning, after two months, and in the end (after 6 months). *SpecH* was composed of 189, 196 and 200 modules in these three measurement times respectively. *SpecA* was composed of 34, 44, and 56 modules respectively.

The specifications are written in formal documents, to enable automatic code generation. Figure 1 shows an example of specification fragment. The first measure we use was size, meant intuitively to represent the number of decisions made in the specifications. Therefore, a size of 1 is given to every simple specified value (such as minimal value and maximal value), and a size of 1 is given to each line of free-text specifications. Therefore, the size of the fragment in Figure 1 is 12 since it has 6 simple values, 1 for the one line of the 'Is Required' specification, and 5 for the five lines of the 'Do on change' specification.

Field Name:	Name
Field Type:	String
Description:	The customer's full name
Minimal Length:	1
Maximal Length:	40
Field Editor:	Text Box
Is Required:	*only **if** the ID field is empty*
Do on change:	***If** the ID field is non-empty, check that it matches the new name. **If** so, enable the 'OK' button, **else** display the 'Name/ID Mismatch' error message.*

Figure 1: Specifications sample

Since the simple values in a specification result in generated code, and hence do not require any coding, the size measure does not reflect the complexity of a given specification for the development team.

Complexity is only created by the free-text specifications – and to represent this, we devised two complexity measures. The first is the *Logic-Based Complexity* that is calculated by counting the number of lines of non-trivial specifications. For the specification shown in Figure 1, this measure would be 6. The second is the *Keyword-Based Complexity* that is inspired by the cyclomatic complexity measure (McCabe, 1976; Watson and McCabe, 1996), in which a sequential method has a complexity of 1, and each decision that causes a split into two directions raises the complexity by 1. This definition is equivalent to defining the complexity as the number of paths in the method's decision graph. We emulate the cyclomatic complexity measure by defining the complexity of free-text specifications paragraphs to be 1 and add the number of appearances of the following popular keywords: if, else, for-every, for-each. For the specification in Figure 1, this measure would be 6 since we count 2 from the 'Is Required' specification (1+1 occurrences of 'if'), and 4 from the 'Do on change' specification (1+2 'if'+1 'else'). Validating with the specifications, we found this emulation as a good and sensible approximation of the actual number of paths in the specification. Although these specifications are free-text, the analysts writing the specifications normally use only these words. They are often manually marked by making the font bold, as shown in Figure 1. This is a project-wide practice, ensuring the quality of data.

Figure 2 presents the logic-based complexity of *SpecH* and *SpecA*, averaged over all modules of each project. As can be observed, the averaged logic-based complexity in the *SpecH* project is four times higher than that of *SpecA*. This difference is important since we expect that both the agile development will continuously simplify *SpecA* (due to continuous refactoring) and that the stabilization phase of the heavyweight development will simplify *SpecH*. In addition, we can see that the average logic-based complexity of SpecH *increased* by 4% during the researched six-month period, while the same metrics *decreased* in SpecA by almost 10%.

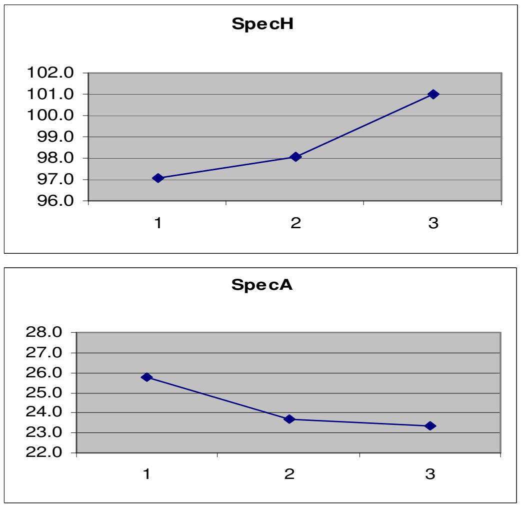


Figure 2: Averaged logic-based complexity in three check points

Figure 3 presents the averaged keyword-based complexity of *SpecH* and *SpecA*. As can be seen, the logic-based complexity and the keyword-based complexity are highly correlated. Also, as in the previous case, in both cases, the values of the averaged keyword-based complexity per each specification in each measurement point are similar, though for *SpecH* the range of values is 3.5 times higher than that of *SpecA*. The trend of change over time is also similar to that observed for logic-based complexity.

This difference in trends over time can be attributed to the different development methods. Note that this difference is for the average complexity over all modules, so the absolute size of each project is irrelevant here. The heavyweight project is in a mature phase; Although 11 modules were added to it during the researched period, the average per-module complexity increased, hinting that most new functionality was embedded in existing modules.

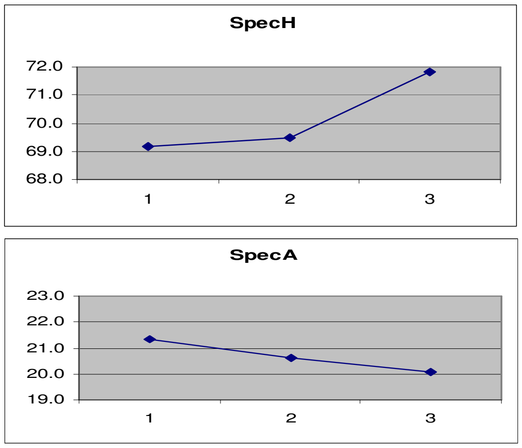


Figure 3: Averaged keyword-based complexity in three check points

In the agile project, on the other hand, 22 new modules were added, and average complexity noticeably decreased. According to the team's testimony, and to XP practices, this is caused by continuous refactoring. When a module gets too complex, it is refactored into (possible several) simpler modules. The goal is to keep the design simple over a long period of time, not assuming the "right" design in advance. In contrast, in traditional projects, the design of modules is usually set in advance. It may also be the case that the high absolute complexity of the heavyweight project, achieved over time (as it is in a more mature state than the agile project), makes refactoring at this stage more expensive and risky.

The difference in absolute complexity can be explained by several factors that do not stem from the development method. For example, the agile project could be inherently simpler than the heavyweight one. From conversations with people in both projects, this is definitely not the case, and two other explanations have been proposed. First, that the agile project reuses more features that are built into the framework, and can be specified in a way that enables automatic code generation. And second, the experience gained from specifying the (earlier) heavyweight project was exploited to specify the agile project in a way that enables greater use of the framework, and less manual coding. Figures 4 and 5 support these explanations, by presenting the ratio of logic-based and keyword-based complexity to the size measure, thereby measuring the proportion of complex to simple specification. The values are about 50% and 20% lower in the agile project, for these two measures respectively.

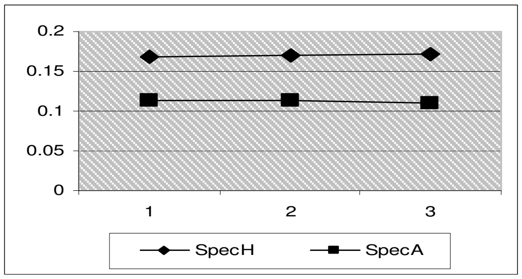


Figure 4: Logic-based complexity to size in three check points

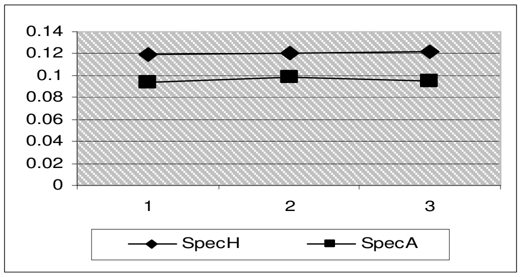


Figure 5: Keyword-based complexity to size in three check points

4.2 The Role of System Analysts

Based on the qualitative data gathered in the interviews, we focus in this subsection on the system analyst role and the changes that characterized it during the transition period. As has been mentioned before, during the transition process, one functional system analyst worked together with the development agile team and another one stayed as part of the external functional analysts group. The group of operational system analysts did not change.

4.2.1. The system analyst role

Following are several expressions with respect to the system analyst role as it was described by the interviewees:

- "A process designer like buildings designer";
- "System analyst is a person who observes a process, understands what the process needs to achieve, and checks how it is possible to improve it";
- "there are system analysts who will finally instruct also how to build the building";
- "[the system analyst] has a global understanding of the system, can analyze the requirements, and can connect the concepts of the operational world to technical concepts";
- "translator from different world of concepts to a system of development concepts".

As can be seen, the role of system analyst is conceived by the interviewees as a central one, both because he or she has a wide perspective at the system and because she or he connects the different parties involved in the development process.

4.2.2. The system analyst role during the transition period

The interviewees also described the change they experienced during the transition process and how they tried to cope with it.

Plotkin describes two main sets of solutions to deal with phenomenon that are characterized by change, and explains how change can be coped with. None of the solutions is exclusive of the others (Plotkin, 1997, pp. 145-152).

The first set of solutions concerns with 'reducing the amount of significant change', thus reducing the change scope. One way to do it is by reducing the period of time between conception and reproductive competence; Meaning, to keep the ratio 'life-span length to numbers of offspring' low, i.e., to maintain high reproductive output in a relatively short period of time. In this case, the change is coped by keeping updated, as far as possible, the genetic instructions of each individual. Plotkin's examples in this chapter are mostly taken from animals' life. The second way to reduce the amount of significant change is to live in a relatively isolated and unpopulated place. A variation of this idea is parents' protection on their offspring by isolating them.

The second set of solutions to cope with the phenomenon of change takes the form of 'if you can't beat it, join it', i.e., change the phenotypes so that they can change with and match the changing features of the world. The first strategy is diversity. One way to accomplish it is to *produce large numbers of different offspring* in order to increase the chance that at least some individuals will be able to face the change. The second strategy, named the 'tracking option', is to give rise to a change within phenotypes, i.e., by *producing phenotypes that change in response to changes in the world.* The tracking option is achieved by knowledge-gaining devices which, according to Plotkin, are the immune system and the intelligent mechanisms of the brain. And thus, the immune system operates in the sphere of *chemistry*, while the brain mechanisms, known as rationality or intelligence, operate in the sphere of the *physical world* of temporal and spatial relationships of events and objects.

In what follows we present some of the interviewees' expressions with respect to the change in the system analyst role during the transition process. The expressions are arranged according to the way the change is coped with.

I. Time aspect

- "I should better understand the constraints because people start to work immediately and I immediately see their side";
- "every two weeks I need to say what will be in the iteration";
- "there is no waste of time".

II. Place aspect

- "everything stays in the customer hands";
- "It helps reduce over spec".

III. Diversity

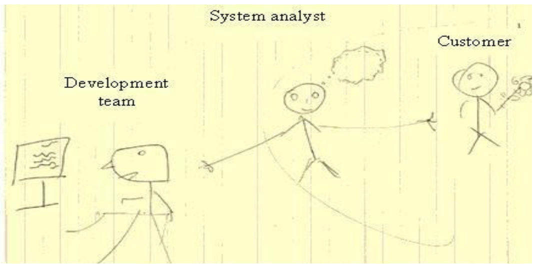
- "everyone is involved and this raises the confidence feeling with respect to the process";
- "there is more interaction".

IV. Knowledge-gaining devices

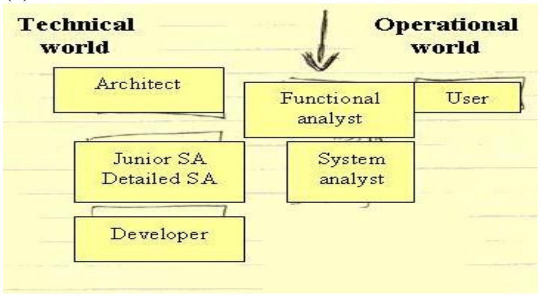
- "XP gave us more power";
- "sometimes we use documents and sometimes only presentations";
- "Explaining the concept, I sometimes see that my concept is wrong";
- "process designer is like building designer".

One of the most salient phenomena that were observed during the interviews was that the interviewees frequently used metaphors and even mentioned this use as a skill that may support the performance of the system analysts' role. The metaphors were diverse and come from different worlds of concepts like buildings, flowers, space ship, flow, journey, and relationship between genders.

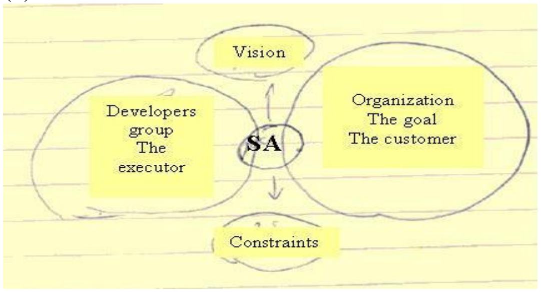
As presented in Section 3, the interviewees were asked to draw the position of the system analyst in the organization as they see it. The three drawings (a)-(c) are presented as part of Figure 6. As can be observed, the system analyst role is mainly conceived as a bridge between the customer and the developers. Specifically, Draw (a) and Draw (c) reflect that this role holder is in a tight middleman situation; Draw (a) reflects that it is not an easy task; Draw (c) reflects also a kind of pressure between the vision and constraints. In Draw (b) the interviewee described the change that was performed in this project in which the system analyst who works on the detailed specifications tends to be part of the development group, the technical side.



(a)



(b)


Vision
Developers group The executor
SA
Organization The goal The customer
Constraints

(c)

Figure 6: The position of the system analyst

We elaborate on how reflective processes can improve a person's understanding of his or her own conception. The importance of reflective processes in software engineering is presented in Hazzan (2002) and in Tomayko and Hazzan (2004) based on Schön (1987). For illustration, we look at Draw (a).

Draw (a) reflects the position of the system analyst in the organization. When the interviewee was asked to reflect on his own draw, he said that the draw illustrates a conflict and that "everything is a matter of explanations". Specifically, he explains that "The customer does not know what is possible to be done. He [the customer] thinks he has a flower. He does not know that he can have two flowers". We conclude with his final words that "Sometimes there is no conflict. Sometimes it is just that customers are from Venus and developers from Mars."

5 SUMMARY

This paper presents the process of transition to agile development in a large-scale software project in the Israeli Air Force focusing on the system analysis and design aspect. Specifically, the project specifications are compared using size and complexity measures and the change in the role of the system analysts is examined. As has been mentioned before, our research continues and further explores the transition process from additional perspectives.

REFERENCES

Beck, K., 2000. *Extreme Programming Explained: Embrace Change,* Addison-Wesley.

Beck, K. with Andres, C., 2005. *Extreme Programming Explained: Embrace Change,* Addison-Wesley, 2nd edition.

Dubinsky, Y., Hazzan, O. and Keren, A. 2005. Introducing Extreme Programming into a Software Project at the Israeli Air Force, *6th International Conference on Extreme Programming and Agile Processes in Software Engineering*, UK.

Dubinsky Y., Talby D., Hazzan O., and Keren A. 2005. Agile Metrics at the Israeli Air Force, *Agile Conference*, Denver, Colorado.

Hazzan, O., 2002. The reflective practitioner perspective in software engineering education, *The Journal of Systems and Software* 63(3), pp. 161-171.

Highsmith, J., 2002. *Agile Software development Ecosystems*, Addison-Wesley.

McCabe, T., 1976. A Complexity Measure, *IEEE Transactions on Software Engineering*.

Mohamed, F., Schmidt, D., and Johnson, R., 1999. *Building Application Frameworks*, Wiley.

Plotkin, H., 1997. *Darwin Machines and the Nature of Knowledge*, Harvard University Press.

Schön, D. A., 1983. *The Reflective Practitioner*, BasicBooks.

Schön, D. A., 1987. *Educating the Reflective Practitioner: Towards a New Design for Teaching and Learning in The Profession*, San Francisco: Jossey-Bass.

Talby, D., Adler, D., Kedem, Y., Nakar, O., Danon, N., and Keren, A., 2002. The Design and Implementation of a Metadata Repository, *INCOSE/IL.*

Tomayko. J. and Hazzan, O., 2004. *Human Aspects of Software Engineering*, Charles River Media.

Watson, A.H. and McCabe, T.J., 1996. Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, *NIST Special Publication 500-235.*