Using the Principles of an Agent-Based Modeling for the Evolution of IS Testing Involving Non-IT Testers

Vineta ARNICĀNE, Guntis ARNICĀNS
University of Latvia, Faculty of Physics and Mathematics, Department of Computing, Latvia

Abstract. Most software relates to information systems that are developed for the needs of companies. Testing of such systems involves the employees of the relevant organization even though they do not have specific IT knowledge (non-IT testers). Non-IT testers are quite ineffective because they don’t know much about testing as such, and because the system that is being tested and the process of testing it are too complicated. This paper proposes usage the principles of agent-based modeling which have been studied and used in the AI world, because they are meant for situations in which the operations of complex systems are to be modeled. Use of the AI techniques can reduce the complexity of testing and evolutionarily improve the efficiency of the testing team’s work.

Keywords. testing, non-IT testers, testing process, agent-based systems

Introduction

A study known as the European Systems and Software Initiative has found that a trend in Europe is that information system software is mostly produced by organizations whose basic area of operations is unrelated to IT. There was stated that 70% of software was produced in that way [1]. The experience of the authors shows that a shortage of qualified IT specialists means that small and medium companies use their own employees as testers of software, because they are going to be the ones who will eventually use the software. Let’s call them non-IT testers [2].

At the same time, testing accounts for between 30 and 60% of all costs related to the software design process, both in terms of money and in terms of time spent [3,4]. This means that the effectiveness of major expenditures is determined by a few experts who are not qualified in the IT sector, but who are knowledgeable about the domain area of the relevant product.

The advantage of non-IT people is that they can have in-depth knowledge about the area of business for which IT software is being designed [5]. Such employees often have years of experience in the relevant area, but they have little knowledge about the technologies that are used in system development of the resulting system.

The issue is how to organize the work of non-IT testers so as to make use of their skills and knowledge as effectively as possible whilst not asking them to do anything that they are unfamiliar with or simply cannot do. How can non-IT testers be brought together into a single team with IT testers who, for their part, are often unfamiliar with
the relevant application domain area? How to observe the constraints of the testing process in terms of limited time and money? How can risks be reduced? This is a complicated task, but it can be simplified if it is modeled as a complex system. In this paper, we propose the use of principles of an evolutionary agent-based modeling in organizing these testing processes.

Complex systems are ones which involve the interaction of a large number of agents. Typical of them is self-organizing collective behavior that is difficult to anticipate on the basis of knowledge about the agents’ behavior, and emergent behavior, moreover, does not result from the existence of a central controller [6].

Many types of agents take part in testing processes – software testers, software designers, users, sponsors, as well as managers from various levels. The goal is the same for all of them – to develop software that is of a high level of quality, is convenient to use, and will function properly for a long time to come. Typically in a testing process, the aim is to find mistakes on the one hand and to make sure about system’s reliability on the other. This is a process that can be structured in various ways – in accordance with testing levels, risk priorities, selected testing techniques, etc. One way or another, however, it is a very creative process in which the individual decisions taken by testers in any specific situation are of fundamental importance. The behavior of testers is emergent. Testers do their work in a creative way, but at the same time they organize it in accordance with management plans, their own experience, their motivation and the way in which they understand the job at hand. As a result of this, their behavior cannot always be forecast or controlled with any precision.

The testing process is one that is naturally evolutionary and self-organizing. As each iteration of testing work comes to an end, it is evaluated – what was good, what was bad, what can be learned, what needs to be kept, what should be improved, what goes missing, etc. Measurements and systems of measurements recorded during the testing process will be of use in this evaluation [7]. The results of the evaluation will point to directions in which the process can move forward and the ways in which participants therein can develop further.

Chapter 1 of this paper offers a brief description of non-IT testers – they testing style and their strengths and weaknesses. Information from [2] is used in this chapter. Chapter 2 addresses the complexity of the testing process if it is viewed from the perspective of non-IT testers and their managers. Chapter 3 offers a brief review of the principles of agent-based system modeling – those which, according to these authors, are appropriate in improving testing processes and making them easier to perceive. Chapter 4 illustrates the way in which the principles of an agent-based system modeling can ease the weaknesses of non-IT testers, making it possible to put their knowledge to better use in terms of finding the way toward new knowledge and skills. Chapter 5 proposes conclusions about the issues that are addressed in the paper, pointing to further research work that can be pursued in the future.

1. Involvement of Non-IT Testers in IS Testing

Non-IT testers are involved in the testing of a company’s information system (IS) in those cases when the company is developing its own system or else is outsourcing the function. Sometimes people from each department in the company conduct testing in parallel to their everyday work. Other times there is an independent and full-time team of testers. In either case, however, these are people with excellent knowledge about the
area of business in which the company is engaged. That means that they can test the IS as experts in the area of the company’s business.

Non-IT testers are good at finding mistakes that are important to users, given that they are going to be the system’s customers and/or users. These are essentially the most important mistakes that need to be discovered during the testing process. Even more, during the testing period these non-IT testers are often the only people who use the system intensively as experts [8].

In depth knowledge about a company’s business allows non-IT testers to do good work at the highest levels of testing, where functional or usability tests must be conducted in terms of acceptance testing and system testing. Even in low-level testing processes, however, there are things which non-IT testers cannot handle because of their lack of IT knowledge and skills. That means that in some cases non-IT testers cannot do all of the necessary work because of one or more problematic areas.

Non-IT testers try to educate themselves about testing procedures, but this is very difficult, because there are few sources of information that would be appropriate for their level of knowledge. Most of the literature requires at least some grounding in IT issues. What’s more, there is often virtually no literature about testing in the native language of the non-IT tester.

Non-IT people conduct the process on the basis of intuition. They usually consider functional and dynamic testing to be the key process. They don’t see an opportunity to conduct testing work in early phases of the software design cycle, when operational software was not yet been created.

Non-IT testers usually have an *ad hoc* testing style, and they are likely to evaluate a software application just from the visual perspective. They use test data which are close to reality, establishing typical situations and chains of activities. They do not find it necessary to test non-standard use of the software, assuming that users of such non-standard elements will know what they are doing.

Popular testing methods include boundary values and partitioning of data in equivalence classes – once again in an intuitive way. If attention is focused on test data, boundary values are examined intuitively. Non-IT testers consider test cases to be only a series of operations, not the relevant data. Data planning is too primitive.

Non-IT testers usually have no experience in working with databases, so they find it difficult to do things such as to examine compatibility among data that are seen on user interfaces and those that are in the database.

Testing processes in which non-IT testers are involved typically have some of the same problems which are seen in testing processes as such - test execution, management of test documentation, measurement frameworks, organization of testing, and the relevant cultural environment [3].

2. The Testing Process and Its Complexity

Organization of testing processes is a particularly difficult problem for non-IT testers. The testing process is very complicated, because it consists of a great many different components which interact amongst themselves, work together, and influence one another. There are several important aspects which have a serious effect on the testing process in terms of determining the testing strategy and plan, as well as the techniques and methods that are to be applied in the testing:

- The amount of time and money that are spent on testing;
− Business area risks, software design and application risks, including the complexity of the business and the software that is being tested, as well as the costs and seriousness of consequences related to mistakes in the software;
− The number of testers, their knowledge, skills and qualifications in the business area and in the matter of testing. This also speaks to the technologies and testing tools that are available to the testers.

Non-IT testers find it difficult to determine the kind of testing that would be appropriate for the specific project, to evaluate the testing results and to plan new testing iteration on the basis of the experience that has been gained.

The testing process is also affected by the methodology for software design that is being used (the waterfall or iterative model, agile technologies, etc.), as well as the technologies that are being applied (the environment, programming language, the database).

If an IT specialist with knowledge about testing joins a team of non-IT testers, then he is usually expected to take over strategic management of the testing process. He tries to make his testing knowledge compatible to the skills and knowledge of the non-IT people. The IT person must be familiar with the testing strategy or method that is being used, and he must be able to explain it to the non-IT people so that the team can put it to use. If the IT person is not familiar with the process, then that automatically means that the entire team lacks the necessary skills.

In analyzing the work of testers, one sometimes has to find that the work has been hindered by something negligible – a small bit of the work which someone has not done because of a lack of competence or skills. That proves to be the reason why the entire job has not been completed, or why the job has not been done at a sufficient level of quality.

The solution here is to divide the task into smaller assignments and to decide which ones each member of the team can handle. For example, let us consider the portioning of data into equivalence classes – something which non-IT testers often do intuitively in their work. Let’s assume that the leader of the non-IT testers or the IT person who has planned the work of the non-IT testers has divided this method into specific steps, decided which non-IT team members can or cannot handle each of the relevant tasks:
1. Stating of boundary values and special values and exceptions in the business.
2. Portioning of acceptable values into equivalence classes.
3. Establishment of equivalence classes for unacceptable values.
4. Design of test cases.
5. Execution of test cases.
6. Evaluation of results of test cases (oracle).
7. Evaluation of overall behavior of software.

When we divide the job up into tasks, we see that the work will require both those people who have good levels of knowledge about the business and those who can say the same about their IT skills.

There are some tasks which the business expert will handle at a better level – defining boundary values, special values and exceptions, as well as serving as the oracle in evaluating the test cases that have been performed (Steps 1 and 6). Someone with good IT knowledge will do better in other areas – establishing equivalence classes for unacceptable values and evaluating the overall behavior of the software (Steps 3 and 7). It can turn out, however, that if the non-IT tester is particularly unskilled in the
performance of the tasks, then the entire application of the method can prove to be a failure.

There are areas in which the level of knowledge is not of decisive importance – design test cases, executing those cases and establishing equivalence classes (Steps 2, 4 and 5). This is because these are tasks that are more of a technical nature.

At the same time, it is clear here that non-IT testers do not need extensive training to perform the 3rd and the 7th task, as well, while someone without knowledge about the relevant business area will have to study much harder to perform the 1st and the 6th task at an acceptable level of quality.

If a larger job is divided up into smaller sub-tasks, and if each of those sub-tasks is delegated to the colleagues who have the appropriate qualifications, then the team can expect to handle the job properly even though each of the individual team members could not ensure that on his or her own. The competences and skills of each team member or the lack thereof becomes readily evident, as do the kinds of competence that should be learned so that the team’s overall work can be improved most effectively.

The team leader or coordinator of the project views each team member as a set of competences and skills. This type of thinking is typical in the agent-based approach. Each team member is a set of agents in which each agent knows how to do something specific, while they all use the same resources – their owner’s brainpower and time.

3. Using the Principles of an Agent-Based Modeling

Agent-based models were established in the development of artificial intellect so as to make it possible to simulate on a computer the natural processes which lead people to work together and to make use of the resources that are available so that specific tasks can be performed in pursuit of the goals that have been defined. Real life is very complicated, and the models can describe that which happens in reality only to a certain extent.

Many books about testing go into great and nuanced detail not only about testing techniques, but also about how the testing process is organized. Sadly, these books and the knowledge described therein are not put to sufficient use. One reason is that these are very long books, and people don’t have the time to peruse them properly. It is also true that the reader can get lots in the details and miss the most important information in the book because of its scope and complexity. This is particularly evident in the situation that is described in this paper – when testing is conducted by people who do not have important knowledge about information technologies and the like.

We propose a simplification of many of the aspects in this process, because we have noticed that people are often far less than effective in dealing with complicated situations. Some do nothing. Others actively do things that are inappropriate. Seldom does an individual move toward the defined goal. If the situation is simpler, and if the individual understands the order of things, then he will feel more comfortable, and the work will be more effective and better targeted. It is also true that people choose those tools which they know how to operate.

We propose a simplification of the situation by differentiating between that which is essential and that which is less essential. This is a gradual process. The first step is to define agent-based models which describe only the most important aspects of the work – after all, it is quite impossible to describe everything which exists in nature. These models, however, will be too complex for the non-specialist, and they are also
meant more for the design of intelligent software. The second step, then, is to choose a few things and principles from the agent-based modeling which are sufficiently simple to understand and which offer a great range of freedom in adapting them to real life. The point is that we thus set out guidelines for the process that is to be performed. In our case, this is testing which must be conducted by personnel who have insufficient knowledge about testing.

Let us now consider the most important concepts and principles that could be used in the testing process, making use in this work of the literature about complex agent-based systems [6, 9-11], particularly [9], as well as sources about testing [4, 5, 12-16].

3.1. Agents

Here is a definition of an agent: "An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives" [10]. In the testing process, the agent is the entire individual or one of the roles that is performed by the individual. By this we mean that the individual performs a simple task which ensures the completion of an elementary process.

During a testing process, an individual can do many different elementary things. He thus represents many agents, all of which, when taken together, establish an agency. The agency of the single individual can itself be seen as an agent. Groups of employees working at a company are also agents, each with its own specific goals and activities. Unlike the agents of a computer system, we have a specific type of limits on agents, as determined by the number of employees. Not all agents of a single type are identical, i.e., they may act slightly differently in terms of functions and consumption of resources.

Each agent will be limited in terms of what can be done, understood and learned, and of the effectiveness with which those things happen. Agents are limited physically (in terms of resources), in the sense of time (the lifespan of the agent, the deadline for completing the work), and in terms of institutional limitations (what is permissible in legal, political and ethical terms).

Agents are divided up into various major classes in terms of their functions. Typical classes include Operator and Manager. Agents of the Operator type do low-level technical work in pursuit of the desired result. Manager-type agents handle organizational and administrative work in managing operators. There is no single ideal organizational system. Each situation will have a different optimal solution - one that depends on concrete agents (or even concrete individuals), on the job, and the surrounding environment, which can be either stable or very malleable and unpredictable.

Let us define a few principles which we consider to be appropriate when organizing testing processes:

- **Principle A1**: We believe that all those who are doing work are primitive agents which can be people (we can assume that the agent is the skill and ability of the individual to do specific things) or software.

- **Principle A2**: Agents are divided up into typical classes so as to set up a hierarchy of subordination and to ensure specialization in the work that is to be done.

- **Principle A3**: We recognize the agents which are embodied in each employee or piece of software so as to learn what we can count on as operations proceed.
- **Principle A4**: We determine the knowledge of each employee or bit of software and the ability of that employee or software to learn new knowledge and skills.
- **Principle A5**: For standardized or anticipated work, we use agent individuals or agent software with lower qualifications.
- **Principle A6**: For non-standardized or innovative work, we use more highly qualified agent individuals or, less often, agent software.

3.2. **Tasks**

The task represents the things that need to be done to achieve global goals, as well as those of individual agents. Tasks at the highest level are usually so complicated that they are difficult even for a good specialist. The “divide and conquer” principle will be of use here – through a process of decomposition, tasks are broken down into pieces until the point where each piece is rather primitive, and it is clear that the task can be handled. There is a hierarchy of sub-tasks here, and various relationships can be applied to the tasks and the sub-tasks so as to establish a graph or network of interdependence.

- **Principle T1**: We break the issue down into tasks and then sub-tasks to the point where each sub-task is one which an agent can handle.
- **Principle T2**: We break down the solution or process of the task into several small activities and operations so as to reduce the complexity of the process.
- **Principle T3**: We define the critical tasks and evaluate risks from the perspective of all stakeholders (client, performer of the work, user).
- **Principle T4**: We identify those critical tasks with respect to which there is a deficit of agents.

3.3. **Operations**

Tasks are handled via operations which make use of available objects. Typical activities with objects include creating them, changing them, destroying them, consuming them and using them. Objects can be data, information, knowledge, tools, storage facilities such as databases, as well as material resources.

It is advisable to choose various typical classes of operations and use standardized templates as to how these classes are to be used. Depending on the behavior of agents involved in the process, operations can be active (the agent does all that is necessary and transfers the results to other agents) or passive (the agent expects other agents to do as much as possible – delivers the entrance objects and receives the resulting objects).

- **Principle O1**: We select a few typical classes of operations and examples of how they are to be used.
- **Principle O2**: We recall that an agent can behave in active or passive terms, and we make sure that the chain of operations does not grind to a halt because agents have been too passive. Agent-software will be more predictable than agent-individual.

3.4. **Coordination and Its Mechanisms**

Coordination can be vertical in the sense of a superior and a subordinate, or it can be horizontal, in which case all agents are equal in their rights. Coordination can be used
to achieve global goals for the entire system, as well as the private goals of individual agents or groups of agents (these, it has to be said, may be out of line with the overall system goal).

Coordination activities are grouped into classes. The class that is known as Strategy speaks to activities which are used to plan the work of the entire system in pursuit of a goal. Supervise refers to activities which oversee subordinate agents. Cooperation refers to activities which ensure that agents not subordinate to one another do things that are advantageous to both agents or to the system as a whole.

Typical mechanisms of coordination include direct supervision (this is a vertical process, with the superior overseeing the work of the subordinate), standardization of work (cooperation is described with precise standards or instructions as to how cooperation and work are to be handled), and mutual adjustment (informal communications).

---

Principle C1: We take steps to monitor the assurance of a strategy and the performance of work in pursuit of the desired goal.

Principle C2: We create opportunities for advantageous cooperation amongst agents, because self-organized systems are far more effective and viable under critical circumstances. We make certain that agents do not get too carried away with their private goals to the point where they start to ignore the overall system goals.

Principle C3: We examine the coordination mechanisms that are present in the system and ensure a balance amongst them, because there can be no ideal coordination mechanism among agent individuals in which each individual prefers his or her own mechanism or combination of mechanisms.

3.5. Organizational Structure and Relationships Therein

In real life, systems are possessed of an organizational structure which defines the groups of elements in the system and the “legal relationships” which they have amongst themselves. At a company, this is a classical organizational structure in which the jobs of employees are defined. There are many different jobs, and the duties of each employee are also different. Jobs can be grouped into classes such as Manager (deals with strategy, organization, delegation of tasks and monitoring of results), Owner (manages resources), and Operator (does the basic work that is needed to create the necessary result).

There can be various relationships among the elements and objects of an organizational structure. Examples of these include producer/consumer (one agent manufactures objects, another consumes them), common limited object (agents need the same object to conduct their operations), report (one agent reports to another), conflict (one agent must report to another agent about a conflict that has arisen), command and instruct (transfer of commands and instructions from agent to another), and delegation (one agent delegates responsibilities and rights to another).

Principle S1: We choose the most appropriate simple model for the actual organization.

Principle S2: If there are many agent individuals in the system, then the relationship among them is already in place and operational. We do not break down the working mechanism, we simply adjust the management of those relationships in which there are problems.
3.6. The Five Capabilities Model

An agent needs a series of capabilities in order to handle the tasks that have been assigned. The agent-based modeling offers the Five Capabilities (5C) model, which has been tested and improved over the course of many years [9]. Five typical groups of capabilities have been described – communications, competence, self, planner, and environment.

Communications ensure cooperation among agents and with the surrounding environment, as well as the maintenance of necessary levels of knowledge. The set of competence capabilities (knowledge and methods) makes it possible to do the technical work that is at hand. Self underpins the agent’s “intimate life”, i.e., agents pursue their goals, tasks and opportunities, they supervise themselves, maintain and improve themselves, and manage their own work. Planner refers to the fact that the agent can take independent decisions on the operating strategy, the way in which tasks will be handled, the order in which tasks will be approached, etc. In other words, agents plan their activities. The set of environment capabilities refers to the ability to obtain information about the surrounding environment, other agents, and ongoing processes.

− Principle F1: We identify the most important capabilities of each agent.
− Principle F2: We identify the most important capabilities that are necessary so as to perform critical tasks;
− Principle F3: We study ways in which missing capabilities can be replaced with existing capabilities, perhaps by choosing different solutions to the problem.
− Principle F4: We determine ways in which agents obtain missing capabilities and make sure that gradually they do, indeed, get them.

4. The evolution of the work of non-IT testers

4.1. The Insight into How the Work of Non-IT testers is Organized

The strength of a chain depends on the strength of its weakest link. The success of a testing job, in analogous terms, depends on the ability to handle the most difficult aspect of the relevant process. The skills of non-IT testers are so negligible and fragmentary that planned and serious testing is not possible. In practice, the situation depends entirely on the issue of whether a good tester is part of the testing team and whether that tester can organize all available resources in pursuit of the goal.

Let us assume that one highly qualified testing specialist is involved in the team. He is charged with improving the testing process. Let us then, at the conceptual level, review the way in which this specialist could use the principles of the agent-based modeling.

1. The situation at the company is determined, and the strategy for what to do is selected (principles S1, S2) on the basis of the existing order of things.
2. He considers employees to be a community of agents (A1), and gradually determines the agents that are available to him, not forgetting about how they cooperate amongst themselves (F1, C3).
3. He plans the testing strategy (C1) and divides all tasks and processes into smaller ones, continuing that process until the tasks and processes are simple and
elementary. This is necessary so that the available agents might understand at least most of them (T1, T2).

4. He determines who can handle each elementary task (A2, A3, A5, A6, F2, F3).

5. He assigns tasks directly to agents or creates conditions which encourage employees themselves to manage “their agents” in pursuit of the necessary goals (A5, A6, T3, T4, C1, C2).

6. He monitors the overall process along with the achievements of each agent (O1, O2, C2, C3).

7. He continually improves the model of activities and creates new agents by training employees or by developing the necessary software (A4, O1, F4).

Because the agent-based modeling that was developed by AI is the cornerstone here, many of the operating paradigms are in place, and they could be adapted without any need to “reinvent the wheel”.

Let us consider a “trivial” testing task which is far more likely to be handled under the proposed approach than it would be if it were just handed over to a non-IT tester without any additional explanations. The task is to review all of the system’s screens by opening and closing them. This task can be divided in two steps - determining a list of all windows and reviewing them.

The list of all windows can be achieved by asking the system developers. The developer can give to tester a list of windows defined in the specification or a list of windows already developed in the system. He can give to tester a full list of windows, but it is possible that he gives just the list with which windows he is familiar. If no list of windows is forthcoming from developers, testers must put together their own lists of windows. The tester can also try to put together a list by determining the desires of future system users, sponsors and technical IT personnel. Certainly the list can be erroneous from certain perspectives.

There are also problems with running the system and opening and closing windows. If the elementary tasks are not defined precisely, then a number of questions like these can emerge: How to start the system under test? How to make sure whether the correct version of the software has been started? How to get to the necessary window? What data to enter, what series of menu choices are to be used, etc.? Does the task of “reviewing windows” really mean just opening a window and then shutting it by any operation. Should all menu choices be reviewed to close window? Should the tester click on icons and shortcuts? How to identify a window? Are the names of windows in tester’s list the same as in the Title part? What is the “window” in the system? Testers may not know how to mark those windows that have already been visited so that they can easily see the percentage of the list that has already been reviewed.

4.2. Reorganization as the Result of the Evolution of the Testing Process

If the testing process is to be improved on the basis of the principles of agent-based modeling, as defined in this paper, it is particularly important to make sure that the changes are evolutionary in nature, i.e., they are natural and gradual, and they slowly move the system toward the condition which is the best in line with the surrounding environment and the global goals and requirements. The environment and the requirements change constantly, and the system must adapt to each new situation, as well. By the word “system” here, we refer to the testing process and all that is related to it at the specific company.
The agent-based modeling defines various ways in which the system adapts to a new situation. Agents can be trained and change their operations. Sometimes older agents “die” and are replaced with new agents with more appropriate capabilities. It is impossible to change people quickly. Changes must be gradual, and successful revolutions of rapid change and destruction of all that has gone before are uncommon in the area of testing, because people have the tendency to return to an older situation which has been comfortable for them.

There is something which most of the aforementioned principles have in common. They seek to reduce the complexity of issues, and they always are focused on greater simplicity. The foundation for this is the essence of the agent-based modeling, which seeks to reduce complexity by decomposition, organization, hierarchy, minimization of interaction, freedom of choice and simple and stable form [9].

In the example which we are considering here, the list of windows is improved as the testers do their work. Initially the tester’s list of windows was probably incomplete and imprecise, no matter how it was obtained. Perhaps not all of the windows provided for in the software specification have yet been created, perhaps unnecessary windows have mistakenly been created, and perhaps neither the software nor its specification lists windows that will be needed by users or clients. The work of the tester will ensure that the list of windows is updated and made more precise. This basically means that testers have learned the method of conformity testing. They can determine whether the requirements are in line with stakeholder interests (nothing is missing, there is nothing that is unnecessary), whether the specification is in line with requirements, and whether the software has been designed in accordance with the specification.

The work of non-IT testers evolves gradually. They get acquainted with the software under test, emerge heuristics in terms of how best to handle each task, acquire new testing methods and techniques.

If the task is “review all windows”, that will also create problems for the manager of testers. If the work of testers is organized on the basis of the principle of functional areas, it can be difficult to know how to divide up windows among testers, because sometimes a single window is used in various contexts, and that means that it will be used in different functional areas. If all testers work precisely with their lists, and no one reviews the system as a whole, then there is the risk that unnecessarily programmed windows will go unnoticed.

The work of the coordinator of testers also evolves constantly. Coordinators learn from their mistakes, develop new heuristics and strategies, gather statistics about resources consumed while tasks are being performed, and adapt themselves to the working style, skills and abilities of testers.

There are two important properties to the testing process – it is capable of gradual evolution, and it is able to organize itself.

5. Conclusions and Future Work

It is simply a matter of reality that many non-IT testers take part in testing processes. They complicate the situation, because thought has to be given on how to divide up tasks and specify the directions of testing so that non-IT testers can handle the job. The testing process can be seen as a complex system. Use of the principles of agent-based modeling means that when non-IT testers are part of the mix, there must be at least one person who knows how to divide up the work so as to make use of the strengths of non-
IT people (their knowledge about the relevant business) and deal with their weaknesses (no technical knowledge about testing methods and information technologies).

It is important that the implementation of the principles of agent-based modeling can be a gradual process in the everyday work of testers. No revolution is needed in the organization of the testing process. Gradually everyone – non-IT testers, IT people, managers – will adapt to the situation, evolve in assessing work and skills, and learning how to make up for skills and abilities that are missing. There is also specialization, however – by functional area, testing method and other skills. Some technical work eventually is automated with the use of tools.

Future research work should focus on other ways of organizing testing processes in which non-IT testers are involved. This process, like testing processes in general, is very complicated, and the authors believe that one of the key reasons for why testing is so often unsuccessful.

Acknowledgements. This research is partly supported by European Social Fund under grant “Doctoral student research and post doctoral research support for University of Latvia”.

References