7

Evolutionary Reduction of the Complexity of Software Testing by Using Multi-Agent System Modeling Principles

Arnicans G. and Arnicane V.
University of Latvia
Latvia

1. Introduction

1.1 A complex nature of software testing

Software testing is a fundamental component in the development of high-quality software. Research shows that 30-60% of the resources that are devoted to software development are focused on testing (Paradiso, 2001; Perry, 2006). Because of this major use of resources and the fact that results that are achieved are often less than adequate, specialists are dissatisfied with the current situation, and they are looking for ways of improving the testing process.

Causes which are often cited in relation to these problems include delays in the launch of testing, a lack of time, a shortage of specialists, and a lack of professionalism in testing. This is due to the fact that specialists from the business with poor IT knowledge are increasingly being put to work as testers, while other IT specialists are not available. There is also a shortage of budget for a proper testing environment, the necessary tools, and the wages of the necessary specialists.

Authors consider that the roots of many problems are found in the complex nature of testing processes. Software testing is a process which can be viewed as a complex system. This allows us to better understand the nature of software testing and to look for more non-traditional approaches toward the restructuring of the process. Our initial goal is to find ways of reducing the complexity of testing. That would make it possible to do the work far more efficiently, to reduce the necessary resources, and to improve the quality of the testing.

1.2 Looking at sociotechnical system by MAS design principles

The testing of software as system is a sociotechnical system. It includes the software, the testers, and the environment in which the testing occurs – the testware, additional software, hardware, and the necessary infrastructure (Joslyn & Rocha, 2000). The main element for the testing is the software system under test, along with the people who are doing the testing.

Testing as system can be considered as a complex system, because it consists of a large number of interacting components (agents, processes, etc.), with a large number of interactions and whose aggregate activity not always is derivable from the summations of the activity of individual components (Sherard & Mostashari, 2009). Complex systems can be described with a multi-agent model (Boccara, 2004; Russel & Norvig, 2003; Shoham & Leyton-Brown, 2009).

Multi-agent models and the architectures which are based upon them are used to design software which, in a simplified way, copies or simulates real-world objects and their behaviour. Designers of models mostly choose issues that are of fundamental importance to complex systems from the real world, and they represent a simplification of those systems.
In testing, as in other complex processes, organisation of the process is of critical importance so that the resources which come from the internal abilities and knowledge of the system can be used efficiently and so that we can adequately adapt these to the surrounding environment. Accordingly, we propose that more attention has to be paid to those organisational principles that are important in establishing multi-agent models to improve the management of testing processes. This approach may help to observe many important issues which we fail to notice in daily routine situations because of the complexity of the relevant system.
A great deal of long-term research has been done to identify the main issues of complex systems and to understand the most important principles on which they operate. The results have served as a foundation for various theories and methodologies which allow us to put together the model of a complex system. This is a simplified model of real life, maintaining only the most critical aspects that are needed to simulate that which occurs in reality. This involves transformation from the real world to an artificial model which can be used for several purposes, for instance, to design software.
We propose more seriously use approach which is based on the usage of knowledge about operation of complex system crystallized in the model as basis of evolving of complex system in the real world. The model covers the most important principles of the real world. If a complex system does not function with sufficient effectiveness, then model that bases on studies of effective systems can serve as an example for the level toward which the complex system should be evolved. The model can show not just the most important elements of the new system, but also undirectly points to unimportant things which the tester should get rid of lest the resources of the system be overburdened. This must be done carefully, however, because it is possible that the model does not take into account some key aspects that are necessary for the full and proper functioning of the system.
The approach toward the improvement of testing processes that we are considering here is just an example of how the principles of multi-agent system modeling can be brought to bear. This could apply to many situations in which it is necessary to address complicated problems related to the active and important participation of individuals. Here is the core question for such research: "How can human organisational principles be used for multi-agent architectures?" We have already noted here that we have chosen the opposite direction: "How can the architecture of a multi-agent system and the principles whereby that architecture is developed be used in order to organise the processes of real-life complex systems in a better way?"
One of the problems here is that there are comparatively few specialists who are familiar with the theories of complex systems and the ways in which they can be modeled. This applies to the modeling of multi-agent systems, as well. Developers or testers will not use methodologies which they don't understand and with respect to which they don't have the necessary skills. This means that the use of MAS modeling principles must be introduced gradually, beginning with the simplest elements. Models meant for the design of intelligent software are usually too complicated for non-specialists in the area of multi-agents. Preference, therefore, must be given to those models which can characterise a complex system or multi-agent system at the conceptual level. The existing situation in the industry,

however, is one in which at least initially, the establishment of the model can be an informal process – even just a mental model in the brain of the person who is organising the testing if he or she does not wish to write it down (Sheard & Mostashari, 2009).

1.3 Managing of system complexity to evolve

A testing process is changing a great deal over the course of time. The software that is being tested is changing, as growing its level of readiness. There are be changes in the testing team, the testing environment, the tools that are brought to bear, the requirements that are applied, and the resources that are available during the period of testing of the newly developed software. This means that testers must constantly adapt to new circumstances by choosing different testing methods and approaches. That is why the testing model must evolve all the time in an iterative sense so that it is in line with reality. For instance, let's assume that we have multi-agent modeling principles that have been chosen and learned well or that have shown that they are no longer of use. In that case, we introduce new principles or supplement the collection of existing ones so that the model is once again in line with the situation at hand.

Because the system is constantly changing and the model has to be adjusted, the system in real life and the model should be more principle-based than rules-based inasmuch as this is possible. This ensures greater freedom for system elements, and the system can operate more effectively and securely whilst, at the same time, reacting in a better way to changes in its surrounding environment (Bar-Yam, 2003). At each moment we must formulate a few principles that allow agents to gain new knowledge and to rearrange themselves in line with the next condition of the system in its planned route of evolution. If testers understand the approach, then it is not necessary for them to be familiar with the model's precise details. It is enough for them to be aware of its most important elements, how they are linked, and how they operate. Formalism becomes important when the need is to replace a human agent with a software agent, as well as to ensure the necessary relationships between people and their computers.

We have observed that the divide-and-conquer principle, which is also known as decomposition, is not used to a sufficient degree. The MAS modeling principles suggest that there be small agents so that their work can be primitive. Larger jobs must be handled by groups of such primitive agents. The techniques, methods and activities of testing can be divided up into many smaller components, which make it possible to use the relevant testing resources more effectively. That particularly applies to the testers who are doing the basic work.

In our approach, the complexity of the testing process is reduced and effectiveness is increased by managing a large number of agents (the skills of employees) and the primitive assignments that are a part of the multi-agent system model. It must be noted that reduction in complexity is a relative concept, because we actually reduce the complexity of only one aspect that is causing problems in terms of the further evolution of the system. In fact, the overall complexity of the system may even increase. The multi-agent model helps us to find weaknesses, and it offers suggestions as to how the situation can be improved – create new agents by training employees, improve planning and co-ordination among agents, or present primitive tasks which can be automated. We believe that testers who are familiar with the most important principles that are described in this chapter can apply them successfully in their work without the establishment of a formal model. From here, we will sketch out ideas as to how MAS modeling principles can help to improve testing processes.

These ideas are based on the practical work of the authors in the field of software testing, including teaching testing processes to others, and organising testing processes.

2. Testing of software as a complex system

2.1 Complexity of software testing

The software that is being tested can be a complex system if it is made up of many autonomous, interlinked and collaborating components or services which are adapting to the environment in which their work is being done, the user who is doing the work, and the situation under which the work is progressing. For instance, such can be systems based on Service Oriented Architecture (Fiadeiro, 2007), as well as component-based software. The software is made up not of large, mutually integrated components, but instead of modular components among which there are many different ways of interaction. The complexity of software is determined not just by its structural or physiological complexity or its size. Also of importance is the social complexity which emerges from the number and intricacy of interactions which involve autonomic components (Fiadeiro, 2007). The tested software can be used in a computer which already has an operating system, with all of the relevant components, and there can be testware or other software used by users during their work. The interaction of all of these elements within a computer is complicated and not always predictable.

Testing processes are handled by people – testers, users and developers. Those who are a part of the testing team also establish a complex system. What is more, testing can involve several overlapping teams – testers, users and developers. There are links between these groups, but there are also links among the people themselves. Some people can be wearing different hats by being a part of different groups.

Testing is a system of systems (SoS) because it includes several complex systems itself – the software that is being tested and the people who are involved in the process. In practice, there are times when several software systems are tested simultaneously to test their mutual interfaces and other types of interaction. In that case, the testing is an even more complex system, because it contains several complex systems in and of itself, and all of the work that is planned and implemented must be balanced to an even greater degree.

The testing system is substantially affected by its environment and by external limitations in terms of the job that has been assigned, the schedule for the work, the budget, and the infrastructure in which the testing system operates – facilities, computers, computer networks, servers, and the like.

2.2 System and environment

2.2.1 Manifestations of system complexity

Complexity as a problem in software engineering is usually addressed by diminishing the complexity of the environment or by increasing the ability of the system to deal with complexity. A third option is complexity engineering or the approach of emergent engineering – using the complexity instead of fighting against it. Appropriate characteristics of complex systems in this regard are self-organizing, co-evolution and emergent behaviour (Heylighen, 2009).

The total complexity of complex systems cannot be described with a single metric. There are different types of complexity (Thorsten et al., 2006) – time, the level of organisation, as well as systemic complexities. Time-related complexities are static and dynamic. Dynamic complexity

refers to the process of the system, the elements, the links among the elements, the number of properties, and changes in differences over the course of time. Static complexity, for its part, expresses these indicators at a specific moment in time. Organisational complexities relate to the structural complexity of the system – the number of elements, the diversity of elements, and the number of links and properties therein. Process-oriented complexity relates to the number and diversity of flows of processes. There can be internal and external systemic complexity. External complexity speaks to the incoming data and resources for the system from the environment which the system can handle and process. Internal complexity refers to the complexity of the model of the system. The boundary between internal and external complexity will depend on the limits of the system itself – what we include as parts of the system and what we leave as elements of the environment (Jost, 2004).

2.2.2 Internal complexity

The elements in a testing system include people, software and hardware. The team includes software testers, software developers and users. Software developers offer consultations to testers about the technological issues of the system and help to produce testware. Users are initially involved as consultants as to the relevant business processes, and later they test it in the system testing and accepttesting levels. The size of the testing team changes over the course of time, depending on the work that needs to be done at any given moment. Only a few users may be involved as consultants at first, while at the level of accepttesting there can be a far greater number of users so as to cut the amount of time that is needed to do the work. In other words, there are dynamic shifts in the structurally organisational complexity of the system (the number of people and links among them), as well as in the process-oriented complexity (the processes in which these people take part and the types of processes that there are).

The software that is being tested changes, as well. New functionality of software are gradually brought into the testing process, found faults are fixed, new requirements are identified and implemented. This changes the number of software modules and services, as well as links among them. This is reflected in the structurally organisational complexity of the system and in the process-oriented complexity thereof.

2.2.3 External complexity

The external complexity of a system is based on new or changed software units – the number and size of modules provided by the developers, as well as the demands from management as to what kinds of testing are expected and how quickly they must be performed. The incoming information and resources have an effect on the internal complexity of the system. If the budget for the system is increased, the elements of the system can be supplemented or changed – new people can be hired and new software can be purchased to improve the testing process.

Demands related to software testing and time limits are of a different nature. These demands change the process-oriented organisational complexity of the process in terms of the testing methods that are necessary and viable, as well as the issue of the scope of the testing – covering all of the software or just a segment thereof. In the latter case, the focus might be on the most critical usage scenarios and the most complicated calculations that are brought to bear.

The external complexity of the system is also based on the complexity of the artefacts which it changes or establishes – mistakes identified in the software, reports about problems, and

documentation such as testing plans, samples, reports on testing and conclusions about the quality of the software that is being tested.

2.3 Organisational complexity of software testing and behaviour of testers

2.3.1 Role of organisation

Testing is a component of the life cycle of information system development projects. A study of several hundred projects of this type in North America found that the greatest effect on project performance indicators such as delivery time, cost, functionality and user satisfaction is had by the structurally organisational complexity of the process, as expressed through the complication and closeness of the various elements in the project's organisational environment, also not forgetting about project resources, support from managers and users, the attitudes of project personnel, and the level of professional skills among the personnel (Xia & Lee, 2004). There is reason to say, therefore, that the level of testing performance can also be affected substantially by structurally organisational complexity.

For that reason, particular attention in this chapter will be focused on ways of changing the organisationally structural complexity of a system by using the principles that are used in the modeling of complex systems.

2.3.2 Perception of system from inside

The complexity of a system depends on the subjective perceptions of the user. The system is viewed by the people who are involved therein. Often they seem just a part of the system, not the entire complexity thereof. It is important to make sure that the part of the system that a user needs to do his or her work is not so complicated from the user's perspective that the work simply becomes impossible.

From another perspective, it can be said that the complexity of complex systems is characterised by emergence, self-organisation, non-linear links among the components, openness and feedback loops (Grobbelaar & Ulieru, 2007).

2.3.3 Reaction of testers to the changes in the environment

Testing is a complex adaptive system. It must react to changes in the external environment and within the system itself.

In practice, it is typical that developers submit software for testing too late, while the deadlines for doing the work are not changed. The result is that testers often have far less time for their work than had been planned, and this will have an effect on the quality of testing. Often enough the work is not done at a sufficient level of quality. Our hypothesis is that if testing is to be more successful, testers must demonstrate skills related to emergence, self-organising, the ability to view synergetic effects, and the ability to handle different tasks related to the process. Then the testing system evolves on the basis of the laws of a complex system.

Testing processes typically have two different kinds of goals – finding mistakes on the one hand and making sure that there are no mistakes on the other. This process is arranged in different ways – in accordance with testing levels, risk priorities, the chosen testing techniques, etc. However, it is always a very creative process in which the individual decisions taken by testers in each specific situation are of great importance. The behaviour of testers is emergent. Testers do their work in a creative way, but they plan and organise it in accordance with management plans, their own experience, their motivations and their level of understanding as to the job at hand. As a result, their behaviour cannot always be predicted and controlled with any great precision.

2.4 Emergent evolving

The testing process can evolve and self-organise in a natural way. When the iteration of each testing process ends, there is an evaluation of what has been good and bad, what we can learn, what needs to be kept, what needs to be improved, and what is lacking. Also of use during the evaluation are measurements that have been taken during the testing and the systems thereof (Chen et al., 2004). The results of the evaluation show directions related to the growth of the process and the development of its participants.

Testing systems are imbued with a series of characteristics that are typical of complex adaptive and evolving systems – self-organisation, emergence, positive and negative feedback, states of equilibrium or absence thereof, the large amount of possibilities, co-evolution, and the nature and history of evolution.

Testing processes are in a stable condition near of the equilibrium when there is no need for new test cases. That usually happens when software is used for a long time without any change in the software or its environment. One or more stably regressive test cases are set up, and these are occasionally used to make sure that the software is continuing to operate in line with requirements. Each time that the software is changed, the testing process loses its condition of equilibrium to a greater or lesser degree, and as new test cases are established so as to stabilise the software, there is once again a permanent set of regression test cases, and the condition of equilibrium is renewed. At the beginning of the testing process, the situation is far from equilibrium, by contrast, and that is particularly true in the early stages of the process, when static analytical testing methods are brought to bear.

Positive feedback about testing processes changes their ecosystem and creates the need for evolution, learning and emergence (Heylighen, 2009). This leads to new versions of software, the identification of new mistakes, the setting out of new goals or missions for the testing, as well as changes in the supply of resources.

There are usually vast numbers of possibilities in testing. There are choices as to strategies, methods, test data and the order in which test cases are assessed. In some cases the method will identify the introductory values that are chosen, although in most cases the value must be chosen from an interval or a list of values.

A very characteristic aspect of testing processes is co-evolution. When one tester teaches another, they both evolve. The former trainer learns to teach others, while the latter person gains knowledge about testing. If a tester finds a mistake made by the software developer, then he gains experience as to how to find the mistakes, while the software developer learns about the mistake and can decide on what to do to make sure that that never happens again.

Testing processes have a history that is based on the situation, chances in terms of what could be done, and what is actually done.

Complexity can be absorbed as the system is adapting to circumstances of the environment and/or evolves.

3. Possibilities to deal with a system complexity

3.1 Exploring of complex systems

People have, for a long time, studied complex systems that exist in our perceived reality. The goals for such research can differ. For instance, there can be a focus on the operating principles of a system so as to:

- Use the principles in another sphere.
- Understand the operating of other complex systems.

- Model and forecast system operations in terms of time and external environmental circumstances.
- Replace the entire system or a part thereof with a technical system such as a computer which uses the relevant software.
- Create new systems which have not existed before.

These examples are based on a study of the general principles which exist in the operations of existing systems, analysis, understanding and the establishment of a model which describes the systems. There are various theories which make it easier to understand and model complex systems. Multi-agent systems are the basis for one such theory.

3.2 Modeling principles of MAS as nature of complex systems

Different methodologies and frameworks have been established on the basis of the multi-agent modelling theory which makes it possible to establish the necessary models more quickly and precisely. Multi-agent systems are based on the systems which make up living organisms, particularly people. The agent can be analogous to a human being, agents conduct the functions that are entrusted to them, they work together, they react to changes in the surrounding environment, and they are born, they die, they educate themselves, and they try to achieve their own goals and the global goals of the entire system.

Our view is that extensive research has made it possible to identify the principles and logic which determine the structure and operations of various complex systems. The identification of the most important things in the real world means that a more primitive model can be used to describe the way in which the system handles a job and exists in the real world while pursuing the mission that has been entrusted to it.

Let us take a look at the primary goal of multi-agent systems. Software that is based on the multi-agent model can intelligently replace an actual person or team in the handling of many different tasks. Here we use the transformation scheme "from the real world to the model", or "software that is based on a model." Our hypothesis is that the transformation can also occur in the opposite direction – "from the model to the real world". This means that we can take a system from the real world and identify the most important principles therein, getting rid of unimportant things that might even be a hindrance in real life. Thus we know the elements and processes of the system which are the most critically important ones – those which determine the results and effectiveness of the operations. We can call these elements the essence of the system.

3.3 Using of MAS models to evolve

3.3.1 Learning from MAS design principles

Complex sociotechnical systems can have different stages of development as determined by their internal structure, organisation, processes and knowledge. We can say that a system is at a higher level of development if it can handle more work or more complicated work at a higher level of effectiveness in terms of the resources that are used. System development usually requires a long time, and it is handled via evolutionary mechanisms. A system can also exist in very different conditions that are dependent on the external environment. One of the key aspects of development and adaptation in a specific environment is self-learning. That is particularly true in the case of systems with little "experience" – i.e., those that are at a low level of development or that are unaware of the best forms of adaptation when there are unexpected changes in the external environment.

Training processes are much quicker and more effective if there are examples from other systems in terms of how to develop the system, the goals that should be pursued, and what to do in various situations that can occur in life. This approach is often the foundation for training about many existing systems. For instance, when we need to improve software testing processes, we can study books which contain information about the experience of others in this regard, as well as recommendations that have proven their validity over the course of time.

We propose a more non-traditional approach to system training and development on the basis of the principles whereby multi-agent systems are developed. If we are familiar with those, we can concentrate on very important issues and speed up the training. We make far more rational use of the resources that are available for the training and for the most typical elements therein. We also can be quite sure that we know the way in which the system will evolve. This is an approach which allows us, in a natural and comparable way, to gain domain-specific knowledge. In our case, that relates to knowledge about software testing.

3.3.2 The evolution of a complex system

When we put together a multi-agent system the plan is that in future it will be changed or will change itself on the basis of new circumstances. If the surrounding environment does not change much and there are no fundamental changes in terms of the requirements that are levelled against the system, then the planned mechanism ensures development and evolution along with changes that occur. If an existing system needs to be changed, however, there is a different approach:

1. We identify the vision and goals so that we know the situation that we want to achieve in terms of the system and its environment.
2. We identify the current condition of the system.
3. We think about strategies in pursuit of the goal and choose the best one.
4. We plan activities in pursuit of the chosen strategy.
5. We do the work in accordance with plans, and we iteratively repeat the whole process from time to time.

In real life, in most cases, complex systems adapt to surrounding circumstances in a gradual and evolutionary way. Revolutionary, major and rapid changes are less common. Revolutionary changes in software testing may occur if the company decides to outsource the testing, as opposed to doing it in house. In that case, a key component of the system has been changed, and links to other external systems must also change (e.g., there must be formal and legal relations between the recipient and the supplier of the testing services).

Let us look at a typical situation in which a system develops gradually so as to ensure a situation that is better for the surrounding environment and for the global goals and demands that relate to the system. In the multi-agent model, we have various ways in which a system can adapt to a new situation. Agents can educate themselves and change their operations. Alternatively, old agents "die" and are replaced by new and more appropriate ones. The operations of agents will also change in accordance with existing knowledge and skills which are the result of a monitoring of changes in the system. This is because the agent may seek to achieve its own goals and those of the entire system with a lesser usage of resources.

The key role of agents in testing processes is performed by people. It is very hard to change people rapidly, and a gradual process is needed instead (Arnicans & Arnicane, 2009). A more revolutionary approach can be taken toward software agents, because the computer

does what it is instructed to do without "thinking." There will be greater problems if software agents become more intelligent, because then they will adopt the shortcomings that relate to human agents. What is more, any revolutionary changes in a complex system are difficult to forecast, because this is a property of complex systems as such.
There are different driving forces behind evolution. There can be planned development in which someone comes up with the correct scenario for development and forces the entire system to pursue it. Another option is to make use of the advantages of a self-adaptive system, which means that the components of the system can be quite free in taking decisions. In that case, the system seeks a status of balance at which it can handle the relevant requirements whilst minimising the resources that are consumed. This approach may ensure an optimal situation at the local level, but it may be that there are other opportunities for implementing requirements even more efficiently. The principles of multi-agent systems make it possible to come up with different strategies for functioning. Most agents may be reactive and only obey commands, but agents can equally be intelligent and proactive in terms of adapting themselves to the situation at hand.

3.3.3 Problem with information entropy

The complexity of a complex system (i.e., a testing system) cannot be stated in absolute measurement units, but it is possible to analyse the characteristics of different complexities during periods of change.
Scientists and practitioners have been dealing with attempts to reduce the complexity of software testing in direct and indirect ways for more than 30 years now, ever since the 1970s. Lots of books and articles have been written about software testing. Initially, the issue had to do with testing techniques, but later authors began to focus more on the organisation and management of testing processes. Our observation is that literature about software testing reflects its complexity and the nature of a complex system quite well. There are many good books for practical use, although each author or group of authors will have different views about testing. The numbers are very different, because they show what the author or authors think about testing and its various aspects.
The excess of literature and the complexity of testing mean the following problems:

- There is no single source of literature which is good enough in terms of demonstrating the essence of testing. Indeed, no such source is possible, because a very complicated system cannot be described in a single book.
- Short books do not offer enough information.
- Excessively thick books sometimes keep readers from studying them fully because they lack time to do so.
- If a reader has absorbed a great deal of literature, then the content can be hard to understand in terms of separating the important from the unimportant. This creates problems in choosing the right strategy or testing technique for the specific situation.

This problem is particularly evident when non-IT specialists become involved in testing as a temporary job that management forces them to do. They usually don't have the motivation to understand the essence of testing, nor do they have the ability to learn about these matters.
In order to change the situation, we must understand the nature of a complex system and the ways in which we can at least reduce the complexity of understanding it.

3.3.4 Design complexity and control complexity

According to Casti, *"... complexity cannot be thought of as an intrinsic property of an isolated (closed) system; it is only made manifest by the interaction of the system with another, usually in the process of measurement and/or control. In this sense, it is probably more meaningful to consider complexity more as a property of the interaction than of the system, although it is clearly associated with both. (..) System complexity is a contingent property arising out of the interaction I between a system S and an observer/decision-maker O"* (Casti, 1986). Here we can talk about the complexity of the system S for the observer O, which is described as design complexity, or about the complexity of the observer O for the system S, which is called control complexity. The two complexities need to be in balance.

If we look for ways of reducing system complexity, we need to understand what a simple system is. Casti (1986) also mentions several characteristics of simple systems:

- Predictable behaviour: We understand the system's behaviour and our ability to forecast its reaction to specific entry data or to the surrounding environment.
- Few interactions and feedback/feedforward loops: The system has a few components among which there few and understandable interactions. The interaction links of the components should not lead to radical changes in the system.
- Centralised decision-making: The behaviour of the system determines by one or only a few decision-makers.
- Decomposability: The system consists of clearly evident components among which there are weak links. Each component is independent in relation to other components.

3.3.5 Simplification of system by MAS models

One of the primary duties in modeling multi-agent systems is to describe the operations of a complex system as simply as possible. Because the basic thought in this is about software design, the recommendation is to use the following techniques to reduce complexity (Booch, 2004):

1. Decomposition: The larger problem is divided up into smaller sub-problems to the point at which the sub-problems can be understood and resolved more easily in isolation from other sub-problems. Complexity is reduced, because each phase in the solution is understood, the solution is simpler, and it can be implemented in a safer and more high-quality way.
2. Abstraction: We use different simplified models to emphasise that which is most important and to hide the details that are not important at the specific level. Complexity is reduced, because we can concentrate on the fundamental aspects of the problem. We can have a conceptual understanding of what is happening, choose the best solution or strategy, and reduce the likelihood of serious mistakes.
3. Hierarchy and organisation: We identify and manage relations among the components which underpin the solution. These can be grouped and seen as a more universal and homogeneous component at a higher level. There are techniques for organising co-operation among components in pursuit of solutions to a complicated issue. Duplication of effort, moreover, can be minimised.

These are effective techniques, but we must keep in mind that when there is a certain level of interdependency in a complex system, the techniques become ineffective. The decomposition of an object is possible only if its behaviour represents a merger of the behaviour of its components. Abstraction can be brought to bear if the description of the object can be prepared independently from other objects in the system. Because a complex

system is basically made up of elements with behaviour that cannot be described as the sum of the behaviour of all of its components, and also because there are elements which cannot be described apart from other elements in the system, these techniques of simplification in complex systems can only be used to a certain degree.

3.3.6 Using of ontologies

Now let us return to the problem of vast and diverse information about software testing. Setting up of multiagent systems can be fundamentally improved if the ontology of modeled system or problem is created. This reduces the complexity of understanding the system, because there are fewer concepts and links among them. It is only natural, then, to hope that the ontology will make it possible to access further and more detailed information that is necessary to deal with the problem at hand. Sadly, no such ontology has been created at this time. That may be a consequence of the complex nature of testing – the ontology would be massive, and it would probably not satisfy all specialists, because each specialist will have a different view of the complex system.

3.3.7 Conservation of complexity

Another opportunity is to reduce the complexity of one part of the system, remembering, then, that the complexity of another part will increase. In a testing system, for instance, it will mean that the complexity will move from testers to software designers, users and managers.

For instance, let's assume that we simplify testing to the point where software is essentially tested only by end users who use it, while even the software developers don't do anything more than elementary unit testing. These users cannot be seen as a part of the testing system in this case, because they are really only software users who encounter various problems therein. As a result, for instance, several million users start to complain about problems to management if it is business software, to the development company or to the development team. This means lots of new links between users and software developers, after which software developers start to look for new processes in approaching each problem, and new processes are used in new versions of the software that is delivered to users. So by simplifying testing we get that the overall complexity of development and use of the software increases. This means that we need to think about ways of affecting the complexity of the system within itself and in the complex system in which our system is a component. A similar situation often exists within a single complex system. When reducing complexity in one aspect, we increase complexity in other aspects.

3.3.8 Reducing of complexity by primitive agents to promote whole system evolving

Our proposal is based on the principle that a complex system – in this case, software testing – teaches itself, adapts to new situations and evolves to the point where it serves its mission whilst using as little in the way of resources as possible. This can be called system evolution, and it means that when we talk about reduced complexity, we are actually thinking about simplifying the system from a fixed perspective so as to ensure its evolution over the course of a longer period in time. When we use the principles of multi-agent systems, we can transfer complexity from one aspect to another.

The overall trend is to ensure that each person who is involved in the testing process has many local and simple views or simple sub-systems which the person already understands.

Each such sub-system must maximally satisfy the needs for simple systems – it consists of simple components that can be decomposed, it has few interactions, there is centralised decision-making process, and the behaviour of this system is predictable. This makes it easier to learn new knowledge and skills so as to do the entrusted job effectively in the context of this small and simple sub-system. The effectiveness of the system may then increase gradually. The overall complexity of the system will certainly increase if it was at a low level before, but the higher effectiveness of the testing process should reduce the complexity of other complex systems that relate to software testing by, for instance, the team of software developers.

4. Using the principles of an agent-based modeling

There are many various frameworks and methodologies that describe the architecture and principles of development of multi-agent systems (Giorgini , 2005).
An explanation of the essence of our approach could be based on any framework that seems acceptable and understandable, because the basic concepts and principles therein are comparable.
We are using the frameworks proposed in (Aart, 2005; Jennings, 2000), because they are based on human organisational principles. Lets us look on the most important concepts that we are holding for our model.

4.1 A Principles based approach

When establishing a model for a testing system and trying to simplify it, we have to keep in mind that it is important to allow the testing system to operate as a complex system, i.e., it is important to create contexts in which they can self-organise to serve our needs without direct design or specification.
Let us take a look at the most importance principles that have been adapted from multi-agent systems and the techniques of establishing them. We can also look at the experience which these authors have had in the area of software testing.
It is hard to define requirements which underpin a complex adaptive system. Far more useful in this regard is a principles-based approach, as opposed to the rules-based approach that is used far more often in describing such systems (Polacek & Verma 2009). Former US Treasury Secretary Henry Paulson has had this to say: "*One important of the IFRS accounting system is that it is principles-based, rather than rules-based. By 'principles-based,' I mean that the system is organised around a relatively small number of ideas or concepts that provide a framework for thinking about specific issues. The advantage of a principles-based system is that it is flexible and sensible in dealing with new or special situations. A rules-based system typically gives more specific guidance than a principles-based system, but it can be too rigid and may lead to a 'tick-the-box' approach. (Paulson, 2006)*" In this case, the IFRS accounting system is a complex system.
The principles-based approach involves a small number of principles related to the specific system in the interest of emergence and evolution. Once the system or the surrounding environment changes, the principles are reviewed, and the set of principles is updated with new ones. The principles for each specific system will differ. The testing system for every piece of software differs from other systems in terms of the job that is to be done, the software that is tested, the knowledge and skills of the testers, and other parameters. We can look at a few examples adapted from (Arnicane & Arnicans, 2008) in terms of sets of principles that could be used by a company which engages in software testing. The

principles therein are sufficiently universal to ensure that they can be used in systems with qualified testers or with testers who do not have special knowledge about software testing and IT. We chose these principles first of all because our practical experience with many projects relates to the organisation of testing projects and to the testing itself. Second, we have made use of our knowledge about complex systems and, particularly, the modeling of complex systems with the help of multi-agent systems.

4.2 Agents

In society, an agent is a person operating in someone's interests (Sterling & Taveter, 2009). In a testing system, that can refer to the tester, the user, the manager or the software developer, all of whom are acting on the basis of the goal of producing high-quality software. The status of an agent can also, however, be assigned to a robot or to software which can operate flexibly and autonomously with the aim of fulfilling its goals (Aart, 2005). We propose to look at a person in a slightly different way. A person is a complex system which can be modeled as a multi-agent system. We could identify agents that are responsible for various primitive parts of the testing process. Other agents within the person need not be identified directly as long as they are not important for the testing process. Thus we can say that the set of agents which represent a person can be seen as an agency at which agents satisfy most of the classical requirements for agents, apart from those properties that apply only to software-agent type agents. The set of human agents who are useful in testing processes can change. The situation improves if the individual gains new knowledge and/or skills about testing methods, for instance.

It can equally be true that human skills among people who are not defined as agents may become necessary in the testing process, and so these people must be identified as new agents. For instance, a tester can be familiar with bookkeeping, and at some point the software needs to be tested in terms of a bookkeeping-related functionality.

At the same time, however, we can look at people not as agencies of independent agents, but instead as a holonic multi-agent system (HMAS). That is because we can say that all external communications occur only through one special agent – the head of the holon. The HMAS has different characteristics that need to be taken into account. When it comes to communications with other agents, for instance:

- Even though the communications pass through a single agent, it is possible to communicate in many different ways (all of the types of communications and the techniques/languages of information transmission which the specific individual can accept on the basis of his or her senses, skills and knowledge).
- The holon may perceive received reports differently in semantic terms than had been intended by the sender.
- The forms of communication that are accepted by the holon may change over the course of time (e.g., people are in different conditions depending on the time of day or night, and there can also be differences in technical resources or in the desire of others to engage in communications).

There are also nuances when it comes to the internal agents of the holon:

- The agent has limited opportunities to do specific work, because the holon is limited – agents tend to operate in a specific sequence of tasks, because there are few people who do different kinds of work simultaneously.
- The work of the agent may be stopped at any time and for an unpredictable and unknown duration.

- It is difficult for the agent to return to work after an interruption if the pause has been very long (people can forget the precise situation in which they were, or it proves impossible to regain the previous condition).
- The agent's results in relation to a single task can differ from one case to the next.
- The agent can submit the results of work that has not been completed.
- The agent can do several jobs at the same time, suspending and then resuming them.
- The reaction of other agents in the holon cannot be predicted if a job is assigned to a specific agent of the holon (the holon involves the emotional and psychological characteristics of a human being).

This means that people, as multi-agent systems, have a dual nature. On the one hand, we can say that we can take different relatively independent agents that can be organised in a new and virtual multi-agent system, but on the other hand, the holon of a person encapsulates all of its agents and determines their availability and the specifics of their use.

If we consider all of the people who are involved in a testing process to be agencies, then we obtain a great many different agents, indeed. This makes it possible to establish a new virtual multi-agent system (VMAS). The more primitive and simple the agents that we have identified in the person, the greater will be the possibility to organise those agents in pursuit of a major task. Now let's look at all of the agents in our VMAS. Several of them will be similar. For instance, the writers of problem reports are agents. Still, each one will be different, because each person will write the report a bit differently. The point is that all agents in a VMAS are unique. We can only assemble groups of similar agents from whom we expect a similar reaction and results.

The quality of results among the agents in a single group can differ very substantially. For instance, an expert will write up a much better problem report than a beginner will do. The agents who come up with the strategy and relevant missions need to keep this in mind before assigning tasks in pursuit of the desired goals. Additional problems for planning agents are based on the fact that available agents are in holons, and that limits the use of these agents at any given moment.

Our hypothesis here is that by using a VMAS, we can affect the complexity in the complex system of software testing. If we simplify the necessary sub-systems, then we can achieve faster and better evolution of the system so that it does its work more effectively and is more likely to adapt to the changing environment. In the context of this hypothesis, we consider a complex system to be software testing, but we also feel that there are other complex systems to which the same hypothesis could apply. Let us look on principles that we propose. For the sake of readability of following explanation we add identifiers to the essential principles, using a different letter for each one according to the perspectives of our model – A (Agents), T (Tasks), O (Operations), S (Structure of organisation), C (Co-ordination), and F (Functioning, which refers to the agent's capabilities).

A1: The agents who will do the work are as primitive as possible. An agent is a person or software that performs a specific task. For instance, agents can be handlers of test cases, evaluators of the results of a test case, the designers of test cases in accordance with criterion C1, documenters of failures that are identified.

A2: Agents are grouped into typical classes or groups, and relations among them or within them are defined. The subordination of agents is specified (leader/subordinate), the upper level specialisation of the agent is defined (operator, manager, planner, resource manager), the lower level specialisation is also defined (evaluator of test results, preparer of

reports about failures, designer of test cases), and the participants which will handle specific tasks (e.g., regression testing, testing of scenarios, testing of performance) are identified.

A3: Agents which exist in a single individual or computer are merged into an agency (holon). According to the A1 principle, a person or computer can contain many primitive agents. Accordingly, the activities of the agents are limited – one agent acts under the framework of the assigned time slot. The parallel activities of several agents, however, can be simulated within the framework of a single agency.

A4: The agencies that are available to us and the agents residing therein must regularly be identified. This means that we are aware of the resources that are available to us, and we can plan specific activities – we choose the operations that can be handled with the available resources, we seek opportunities to gain a new agency with a necessary agent, or we establish a new agent in the existing agency.

A5: We determine the ability of each agency to create new agents. We must be familiar with the ability of each employee to gain new skills (i.e., create a new agent in himself). We must also be familiar with the computer and its software in terms of opportunities to use or configure these or to create opportunities for automatic adaptation of the software).

A6: Planning the effective use of the agency. Each agency has its operating costs. We use employees with a low level of qualifications to handle primitive and standardised tasks. The computer is what handles operations that are frequently repeated and can be computerised – this makes the testing automatic. Highly qualified employees must be used for non-standard and innovative operations, and they should not be assigned tasks that can actually be handled by employees who are lower on the ladder of qualifications.

4.3 Tasks

A task or action is something done to achieve the aims of global and individual agents. Higher-level tasks are usually so complicated that even fine specialists have problems in handling them. These are simplified via decomposition. Tasks are divided up into sub-tasks to the point where they become quite primitive and it is clear how they can be handled or, alternatively, that they cannot be split up any further unless the quality of the process can be lost.

Decomposition leads to a hierarchy of the tasks that must be done. Different links can be made among the tasks and sub-tasks to create a graph or network of dependency.

Task related principles are following.

T1: We divide up the tasks to get primitive sub-tasks. There must be harmony here with the A1 principle. The more primitive the tasks and the agents that handle them, the less complex the system will be. We assume that the system's "complexity of understanding" declines more rapidly than the "complexity of the quantity of components" increases, because we can make use of resources that are offered by abstraction and grouping. Here we have great opportunities for optimal operations, because major tasks can be handled by more than one agent.

T2: Determination of critical tasks. In evaluating risks and the interests of various stakeholders (the client, the agent doing the work, the user), we can prioritise the tasks that are necessary. We start this evaluation by the highest-level tasks. Evaluation of sub-tasks is conducted only for the critical upper tasks. This helps us to define our testing strategy, to decide whether new tasks must be created, and to come up with conditions for the establishment of a plan to perform the tasks.

T3: Defining those groups of tasks which will require a lot of time to perform. Here we determine which tasks are interdependent on the basis of various criteria that will affect the total amount of time that is needed (the tasks have to be handled in sequence, they consume one and the same resource, etc.).
T4: Determining those tasks which only a limited number of agents can handle. Usually there will be tasks which require someone with a high level of qualifications to handle them, and that means that there is a deficit of appropriate agents and agencies. Such agencies must be reserved for these critical functions, keeping them from doing other, simpler work.

4.4 Operations

The operations are handled by agents in order to fulfill the task. Operations use objects that are available, for instance, data, information, knowledge, tools, data bases, or material resources. Typical operations with objects are creating, modifying, destroying and consuming of them. Let us also note that operations can be described with a precise algorithm, or they can be also ones in which the agent must come up with its own innovative solution in each specific situation.
A typical operation in a testing system is the handling of a test case related to the software that is being tested. The tester initiates the work, ensures the necessary data, receives the results, and then compares them to correct result produced by the oracle to see whether the output data are correct. Operations related principles:
O1: We divide up the operation into primitive sub-operations. As was the case with principle T1, we reduce the complexity of the overall job and make it more possible to manoeuvre with the selection of agencies for each sub-operation. It is also easier to monitor the performance of the work.
O2: We define the most important classes of operations and specify their operational algorithm. Typical solutions are identified for those operations that are more important and must be handled more often. Templates help us to describe the way in which the operation is to be conducted. Those operations that can be performed on the computer can later be programms, and the relevant software agents can be created.
O3: Protection against deadlock. Because most agencies will be human beings who have a great deal of freedom in taking decisions, we must make sure that there are controls to ensure that the work is done. In practice, an agency can make some of his agents passive or fail to give them the time that is necessary to do the work. The result is that work on an operation can come to a halt, and that will have an effect on the behaviour of other agencies. When the agent is actually software, the work is easier to adjust and forecast.

4.5 Organizational structure and relationships therein

Testing jobs can often be handled more quickly and successfully if agents handle them not alone, but in partnership with other agents. This means teamwork among agents. A new team of agents can be set up for every task.
Here is what is typical for teamwork in multi-agent environments (Dunin-Kepli & Verbrugge, 2010):

- The agents work together in pursuit of a common goal.
- They monitor the progress of the group's work.
- They help each other as necessary.
- They co-ordinate the work of agents so that they do not hinder each other's work.

- They analyse and discuss successes and failures because that helps to improve the work that the team does.
- They do not compete amongst one another in pursuit of the common goal.

Organizatorial structure related principles are following.

S1: We select the best organisational structure for each class of tasks. We have an official organisational structure for our agencies, and that must be taken into account. At the same time, however, there are also informal relations among agencies. We can choose and govern both formal and informal structures. This must be in line with principle C1.

S2: We use existing organisational structures. The analogue is with principle C3.

4.6 Coordination and its mechanisms

Each team has certain co-ordination mechanisms. Coordination can be vertical with the leader and subordinates or horizontal when agents have equal rights. Coordination can be implemented depending on the environment, the activities that are to be pursued, and the organisational structure that is at hand:

- Direct supervision underpins vertical co-operation, with the manager overseeing the work of subordinates.
- Standardisation of work, where co-operation is based on precise standards or instructions as to the co-operation and the work that is being done.
- Mutual adjustment, with agents agreeing among themselves on their co-operation without any encouragement from the outside.

There are organizatorial structure often established in company determining groups of system's elements and their „legal interactions". Like each employee has its position in company with his duties and rights, each agent also acts in some position or role.

Coordination related principles:

C1: We specify the best possible co-ordination mechanism for each task. Depending on our strategy for ensuring the testing process and the agencies that are available, we define the best co-ordination mechanisms among agents and among agencies. The testing process is very flexible and dynamic. It depends on the project phase and the testing methods that are used. This means that many different versions of co-operation will be used simultaneously in the system.

C2: Promoting co-operation among agents and agencies. We set up opportunities for co-operation, show why they should be used, ensure an environment for the pursuit of global goals, and then let agencies themselves decide on co-operation as such. The goal could be to set up a self-organising system, because such a system is far more effective and viable under critical circumstances. Let's be careful, however, to make sure that the agencies do not get too carried away with private goals and ignore the system's goals.

C3: Use of existing co-operation mechanisms. The cornerstone for the testing process, at the end of the day, will involve living people, and the organisation will have specific co-operation models for specific individuals. There is no ideal co-ordination mechanism among people, because each person prefers his own desired mechanism or a combination of mechanisms. This will depend on the individual's personal characteristics, the level of the individual's maturity, and the goals which the individual sets. People don't like to accept rapid changes in their lives, and that's why we need to try to use the existing co-operation model, gradually transforming it in the desired direction.

4.7 Capabilities

An agent has to have necessary capabilities in order to handle the tasks that are assigned to it. We have chosen the Five Capabilities model for the modeling and management of the abilities of agents (Aart, 2005). The capabilities grouped therein include communication, competence, self, planner, and environment. These are the most important considerations if the agents are to be a capable as possible in handling tasks in our system.

Communication ensures co-operation between one agent and others, as well as with the surrounding environment and the maintenance of the necessary knowledge. *Competence* (knowledge and methods) ensure that the job can be done in technical terms. *Self* supports the agent's "intimate life" – the agent maintains its goals, the work that needs to be done and the opportunities that are at hand, it supervises, maintains and improves itself, and it manages its operations. *Planner* refers to the ability of the agent to decide on operating strategies, the order in which tasks are to be handled, what techniques are to be used, etc. – in other words, the agent plans its own operations. Finally, the capabilities under the heading of *environment* enable the agent to gain information about the surrounding environment, other agents, and the processes which are occurring.

Capabilities related principles are following:

F1: We determine the most important skills of agents and agencies. We must know the resources that are available to us before we can plan our testing strategy and activities.

F2: We determine the most important skills of agencies that are needed for the most critical tasks. This has to be harmonised with the results that we get when applying principles T1, T2 and T3. We can define the missing skills, which will be the difference between those abilities found with principle F1 and those found with principle F2.

F3: Seeking out alternative skills. We look for ways of replacing those skills that are missing with others, perhaps looking for entirely different solutions to the problem. Testing is a process in which different methods can be used to achieve the same goal, and that also means different functions. This represents dynamic adaptation to the circumstances which prevail.

F4: Developing new skills. We look at ways of ensuring those skills which are missing and ensure that they are gradually developed. This means creating new agents by training employees, obtaining new software, or configuring existing software.

4.8 Evolving by choosing the other principles

The most important prerequisite in the application of principles in practice is to do so gradually and moderately. First choose some principles which you understand and believe can be implemented without much difficulty. After awhile, these may become principles that are automatically understood. Use them until they are not longer actual according to the new circumstances. When some principles prove to be of no more use, replace them with others. Thus the testing team gradually learns about the basic principles and evolves in its work. If the choice of principles is repeated in a cyclical way in support of further development, then it is necessary to review other general principles related to complex systems and to choose the appropriate ones. For instance, typical principles for the establishment of complex systems can be found in (Polacek & Verma, 2009). It is also important to take into account domain-specific principles. For software testing, basic principles can easily be adapted from (Kaner et al., 2002), for instance. That is a source which offers several hundred principles and recommendations that are important for testing, have been examined in practice, and can be used successfully in combination with MAS modeling principles.

5. The evolution of the work of testers

5.1 A lack of qualified testers

In 2000 study by the European Systems and Software Initiative found that 70% of all software has been designed by organisations which do not specialise in software design (Haugh, 2001). There is no reason to think that this share has dropped. The organisation particularly looked at problems related to the quality of software and to testing. It found that testing was not being done at an adequate level. The main cause, according to the specialists, was a shortage of good testing specialists. Instead, there was intensive use of employees with good business knowledge, but poor knowledge about IT in general and software testing in particular (Marick, 2001). Problems related to the intensive use of non-IT testers are described in (Arnicane, 2007).

Let us look at a fairly typical situation which the authors have encountered at non-IT companies. Testing is often organised on the basis of a simple and understandable principle – each functional part of the software involves a tester who has to test that particular area. There is a fundamental shortcoming here, however. A tester without proper qualifications usually limits the work just to typical test cases, and that is usually not effective. Sometimes nothing in this regard changes over the course of many years.

5.2 Towards MAS paradigm

Companies are not satisfied with this, and so they bring in one or more testing specialists. A good specialist can plan a more or less optimal testing strategy on the basis of his capabilities and knowledge. Alas, there is a lack of human resources to pursue these plans. An unqualified tester cannot be assigned a high-level task, because he simply will not know what to do with it.

Testing is much improved if the activities are divided up into atypically small sub-activities that are comparatively primitive. Small and understandable tasks are delegated to non-IT testers, with the necessary individual training that can usually be completed in a short period of time. The testing professional handles only those primitive tasks which are nevertheless too complicated for non-IT testers. What's more, the specialist basically has to deal only with strategic planning, the detailed delegation of tasks, training, and monitoring of the work that is done. The professionalism of non-IT testers gradually increases, they can be given less detailed tasks, and the effectiveness of the total team is enhanced. When fundamentally different or new circumstances occur, the activities are once again split up into a greater number of parts, and training begins anew.

This process can be described very well with multi-agent system models. The advantage of the multi-system approach is that the same principles can be used to describe the co-operation or symbiosis between people and computers in pursuit of common goals. This makes it easier to replace computer operations with human work, as well as to formalise operations, describing them with algorithms so that it becomes possible to establish software-based agents.

5.3 A Sample of strategy to apply the MAS principles

Let's assume here that a company has found a good testing specialist. At the conceptual level, let us see how the aforementioned principles are used to restructure the software testing. We'll list the strategic activities which yielded positive results in real projects. In line with MAS principles, these can be handled simultaneously and in parallel. Each activity is

pursued until such time as the internal condition of the system or the surrounding environment has changed:

- When a new person (agent) arrives at the company, the organisational situation is determined along with relations among employees so as to better integrate into the company and to make changes gradually and without major revolutions (S2). We look at the organisational structures that are best for reaching each fundamental testing goal, i.e., at how to work with agents more effectively (S1). We must assess the group of testers (the interior part of the system), as well as the company's other departments, partners, and users of the tested software (the environment for the software).
- We consider employees to be holonic agents who work together with other agents (A3), and we gradually identify the available people and their abilities, as well as the software, its functionality (A1, F1), and the way in which they work together (F1, C3). Accordingly, we can identify the testing tasks that we can achieve, and in the case of the simplest tasks, we can set deadlines and determine necessary levels of quality. We can plan the testing strategy, as well as a strategy for further training.
- We plan the testing strategy, and we divide up all of the tasks (T1) and processes (O1, O2). We divide them up into tasks and processes that are as small, simple and elementary as possible, the aim being to make sure that the available agents can handle them. We reject work that cannot be done, or we look for new and necessary agents to do that work.
- We determine which agent can handle each elementary task (A2, A3), judge whether quick training is needed (A5, F2, F4), assess the speed and quality of the work (A6, T1, T3), and determine the order of the tasks and the deadline for completing them (T1, T2). In a worst-case situation, we initiate training of employees to create new agents, or we look for ways of doing the job in a different way (F3).
- We assign the work to agents directly or create conditions in which employees are motivated to do the necessary work on their own (C1, C2, C3).
- We evaluate the risks which are associated with the most important tasks (A4, A5, T2, T3, T4, O3) and facilitate restructuring of the testing process if the performance of important tasks is endangered.
- We monitor the overall process and the achievements of each agent (O2, O3, C2, C3).
- We constantly improve the operating model and create new agents by training employees or designing appropriate software (A4, A5, T4, O2, F4).

5.4 Obtaining of New Skills and Agents

If the system is evolving more quickly, it is critically important to make sure that there are at least a few capable agents and that at least one of the agents has the necessary critical knowledge. Otherwise the system will be developed slowly, or it may not develop at all. In that case, there is the risk that as circumstances change, the system will not be able to handle even its most basic functions. In this case, the professional testing specialist who is brought into the process will require many different skills and areas of knowledge.

It is possible to organise an increase in the number of agents by hiring new employees and training them or by training existent employees - testers. Here is the technique that should be used for training: The future agent should first be allowed to handle the task as best he or she can. Then the work is corrected or completed by a professional agent, after which the trainee compares his or her results to the work of the professional agent. That helps people to learn.

5.5 A complexity for the key testers

Evolution of the system is achieved by reducing the complexity for the majority of employees who are involved in the testing, because that gives them a better understanding of the work that is done. They only are assigned tasks which they can achieve, they constantly learn new skills, and co-operation is basically informal in the context of a higher-level task, as opposed to being subordinated to the official structure of the company. The overall structure of the process does not disappear, however, because the complexity of the work of the professional tester is increased substantially. The tester is forced to devote more time to planning, training and organisational aspects of the work, as opposed to the testing process itself. One such employee can work with a small number of non-IT employees. Otherwise, the speed of development diminishes substantially.

6. Conclusions

There is discussed software testing as a complex system and considered ways of reducing its complexity in this chapter. Complexity is an inherent characteristic of complex systems, and it is not, therefore, possible to reduce their overall complexity. It is, however, possible to reduce the complexity of individual aspects of the system so as, for instance, to ensure that the work of people who are involved with the system becomes simpler, easier and more likely to be handled in a high-quality way.

A set of principles is offered based on ideas from multi-agent modeling methodologies and author's expierence in software testing. By gradually putting these into practice, it is possible to ensure that the work of the testers gradually evolves because the complexity of the testing is reduced insofar as many of the people in the process are concerned.

In order to model testing system there are used the basic concepts of multi-agent systems – agent, task, operation, co-ordination and organisational structure. Traditionally, people are modeled as agents. It was handy to perceive people as complex systems with agents as elements that are responsible for various primitive aspects of testing processes. Other agents are not identified within people as long as they are not proven to be important as participants in testing process. Each testing job is done by one or more agents. Agents can belong to one or more individuals who are doing the testing work. This work is based on principles from the theory of modelling multi-agent systems. Testing systems are very different – by SUT, testing team, constraints, priorities and accordingly by sets of principles. There are considered those principles in this chapter which relate to improvement of the management of testing processes, because it is one of the ways how to achieve improvements in testing process with relatively small effort.

There are not discused principles which could be used to ensure adequate and effective testing in this chapter, for instance, a minimal set of test cases that can be handled with minimal effort, that have results that can be evaluated precisely, and that allow for the conclusion that the handling of the consequences of remaining potential errors could be cheaper than the resources that would be needed for additional testing.

The sense of testing as a complex system allows explain why there have been so many failures in practice in this regard – money and time have been expended, work and effort have been invested, but the result is not achieved in that there remain numerous problems in the software under test. The errors are found as the software is used. The complexity of

testing systems means that it is hard to evaluate the effort, time and money that will be needed for the work when the testing process is being planned.
Another result of the fact that testing is a complex system is that it is basically not possible to define or to describe the work of testers with procedures and then control the compliance of the work that is being done with these procedures. It is necessary to allow testers to ensure emergence, self-organisation and flexible behaviour that will lead allow them to deal with the situation at hand.
Historically, the complexity of testing systems has been limited by limiting the freedom of its elements and ensuring as much order as possible. This is done by implementing limitations such as the demand for plans, the strict adherence to the plans, and the observance of written procedures related to the work. In many cases, however, there can be plans, reports and a lot of bureaucracy, but the results will nonetheless be far from perfection. Perhaps that is because of all of the major limitations. Testing is a creative process.
Further research is needed into how an environment can be set up for a testing system in which it can make use of all of its advantages as a complex system – the ability to deal with complicated tasks in a creative way whilst, at the same time, not complicating the work of others who are involved in software development - developers, users and managers. Testing which is highly restricted by procedures and rules is more advantageous to management, because it is more predictable, and it is easier for managers to reject the idea that they are responsible for failures. In such cases, however, testing is no longer a complex system, it is just a complicated one. The greater the level of freedom in the elements of the testing system, the more the testing system behaves like a complex system which is harder to understand, describe and predict. Research is needed into ways of balancing risks that come from the unpredictability of the system with the benefits that the complex system provides.
Even though it seems that there is no real chance to set up formal models at this time, it is worth looking at whether there cannot be special tools and methodologies which make it easier to observe the principles of establishing a multi-agent system in testing processes.
In conclusion, it has to be stressed that the use of the ideas described in this chapter will largely depend on whether the testing team has at least one specialist who understands fundamentals of multi-agent systems and has good knowledge about software testing. There are no empirical data collected yet whether there can be fundamental improvements to testing processes if this is not the case.
The critical need is for a specialist who can imagine the testing system as a set of many primitive agents which engage in small tasks in pursuit of the overall goal, as well as can establish and constantly renew the concrete model for the specific project at least in a mental and informal way, the goal always being to allow the testing system to evolve gradually toward reduced complexity, at least insofar as testers who are involved in the system are concerned.

7. Acknowledgement

This work has been supported by the European Social Fund Project No. 2009/0216/1DP/ 1.1.1.2.0/09/APIA/VIAA/044. We wish to thank Karlis Streips for improving the English of this chapter.

8. References

Aart, C. (2005). *Organizational Principles for Multi-Agent Architectures*, Birkhauser Verlag, ISBN 3764372133, Basel Boston Berlin

Arnicane, V. & Arnicans, G. (2008). Using the Principles of an Agent-Based Modeling for the Evolution of IS Testing Involving Non-IT Testers, *Proceedings of 8th International Baltic Conference on Databases and Information Systems (Baltic DB&IS 2008)*, pp. 129-140, ISBN 978-9985-59-789-7, Tallinn, Estonia, June 2008, Tallinn University of Technology Press, Tallinn, Estonia

Arnicane, V. (2007). Use of Non-IT Testers in Software Development, *Proceedings of 8th International Conference on Product Focused Software Process Improvement (PROFES 2007),* pp. 175–187, ISBN 3-540-73459-7, Riga, Latvia, July 2007, Berlin: Springer, (LNCS), Riga, Latvia

Arnicans, G. & Arnicane, V. (2009). Opportunities to Improve Software Testing Processes on Basis of Multi-Agent Modelling, In : *Databases and Information Systems V: Selected Papers from the Eighth International Baltic Conference, DB&IS 2008 - Volume 187 Frontiers in Artificial Intelligence and Applications,* Haav H. M., & Kalja, A., (Ed.), 143-156, IOS Press, ISBN 1586039393, Amsterdam Berlin Oxford Tokyo Washington DC

Bar-Yam, Y. (2003). When systems engineering fails-toward complex systems engineering, *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, Vol.2, pp. 2021-2028, ISBN: 0-7803-7952-7, Oct., 2003, USA

Boccara, N. (2004). *Modeling Complex Systems,* Springer, ISBN 0-387-40462-7, New York Berlin Heidelberg Hong Kong London Milan Paris Tokyo

Booch, G. (2004). *Object-Oriented Analysis and Design with Applications,* Addison Wesley Longman Publishing Co., Inc., ISBN 020189551X, Redwood City, CA, USA

Chen, Y., Probert, R. L. & Robeson, K. (2004). Effective test metrics for test strategy evolution, *CASCON '04: Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pp. 111-123, ISBN , Markham, Ontario, Canada, October, 2004, IBM Press, Markham Ontario Canada

Dunin-Kepli, B. M. and Verbrugge, R. (2010). *Teamwork in Multi-Agent Systems: A Formal Approach*, Wiley, ISBN 0470699881

Fiadeiro, J. L. (2007). Designing for Software's Social Complexity, *Computer*, Vol. 40, No. 1, (Jan. 2007), 34-39, ISSN 0018-9162

Giorgini P. (2005). Agent-Oriented Methodologies: An Introduction, In: *Agent-oriented Methodologies,* Henderson-Sellers B. & Giorgini P., (Ed.), 1-19, Idea Group Publishing, ISBN 1591405815, Hershey London Melbourne Singapore

Grobbelaar, S. & Ulieru, M. (2007). Complex networks as control paradigm for complex systems, In: *Proceedings of IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 4069-4074, ISBN 978-1-4244-0991-4, Montreal, QC, Canada, Oct. 2007

Haug, M. (2001). Software Process Improvement: A European View, In: *Software Quality Approaches: Testing, Verification, and Validation*, Haug, M., Olsen, E. W. & Consolini, L. (Ed.), 3-14, ISBN 3540417842, Berlin Heidelberg New York Barcelona Hong Kong London Milan Paris Tokyo

Heylighen F. (2009). Complexity and Self-organization, In: *Encyclopedia of Library and Information Sciences,* Bates, M. J. & Maack M. N., (Ed.), CRC Press, ISBN 084939712X

J. Casti, J. L. (1986). On System Complexity: Identification, Measurement and Management, In: *Complexity, Language and Life: Mathematical Approaches,* Casti, J. L. & Karlqvist A., (Ed.), 146 - 173, Springer-Verlag, ISBN 3-540-16180-5, Berlin Heidelberg NewYork Tokyo

Jennings N. R. (2000). On agent-based software engineering. *Artifical Intelligence*, Vol. 117, No. 2, 277-296, ISSN 0004-3702

Joslyn, C. & Rocha, L.M. (2000). Towards semiotic agent-based models of socio-technical Organizations, In: *Proceeding of AI, Simulation and Planning in High Autonomy Systems (AIS 2000)*, pp. 70–79, Tucson, Arizona, USA

Jost, J. (2004). External and internal complexity of complex adaptive systems, *Theory in Biosciences,* Vol. 123, No. 1, (June 2004), 69-88, ISSN 431-7613

Kaner, C., Bach, J. & Pettichord B. (2002). *Lessons Learned in Software Testing: A Context-Driven Approach*, Wiley Computer Publishing, ISBN 0-471-08112-4, New York Chischester Weinheim Brisbane Singapore Toronto

Marick, B., (2001). Classic Testing Mistakes, In: *Software Quality Approaches: Testing, Verification, and Validation*, Haug, M., Olsen, E. W. & Consolini, L. (Ed.), 57-82, Springer-Verlag, ISBN 3540417842, Berlin Heidelberg New York Barcelona Hong Kong London Milan Paris Tokyo

Paradiso, M. (2001). Software Verification & Validation Introduced, In: *Software Quality Approaches: Testing, Verification, and Validation*, Haug, M., Olsen, E. W. & Consolini, L. (Ed)., 36-45, Springer-Verlag, ISBN 3540417842, Berlin Heidelberg New York Barcelona Hong Kong London Milan Paris Tokyo

Paulson, H. M. (2006). Remarks of Treasury Secretary Henry N. Paulson on the competitiveness of U.S. Capital marlets Economic Club of New York, retrieved September 20, 2010, from http://www.ustreas.gov/press/releases/hp174.htm

Perry, W. E. (2006). *Effective Methods for Software Testing*, Wiley Publishing, Inc, ISBN 0-7645-9837-6, Indianapolis, Indiana

Polacek, G. A. & Verma, D. (2009). Requirements Engineering for Complex Adaptive Systems: Principles vs. Rules, *Proceedings of the 7th Annual Conference on Systems Engineering Research CSER 2009*, ISBN 978-0-9562440-0-0, Loughborough University, UK, April 2009, Research School of Systems Engineering, Loughborough University, Loughborough, UK

Russell, S. J. & Norvig, P. (2003). *Artificial Intelligence: a Modern Approach*, Prentice Hall, ISBN 0-13-080302-2, London Sidney Singapore Hong Kong Toronto Tokyo New Jersey

Sheard, S. A. & Mostashari, A. (2009). Principles of complex systems for systems engineering, *Systems Engineering,* Vol. 12, No. 4, (Nov. 2009), 295-311, ISSN 1098-1241

Shoham, Y & Leyton-Brown K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*, Cambridge University Press, ISBN 0521899435, Cambridge New York Melbourne Madrid Cape Town Singapore Sau Paulo Delhi

Sterling L. & Taveter K. (2009). *The Art of Agent-Oriented Modeling,* The MIT Press, ISBN 978-0-262-01311-6, Cambridge, Massachusetts London, England

Thorsten, P., Bose, F. & Windt, K. (2006). Autonomously controlled processes – characterisation of complex production systems, In: *Proceedings of 3rd International CIRP Conference on Digital Enterprise Technology (DET),* Setubal, Portugal, Sept. 2006

Xia, W. & Lee, G. (2004). Grasping the complexity of IS development projects. *Commun. ACM,* Vol. 47, No. 5, (May 2004), 68-74, ISSN 0001-0782