

## Using the Sponsor-User-Programmer Model to Improve the Testing Process

**Guntis Arnicans, Vineta Arnicane**

University of Latvia, Raiņa Blvd 19, Rīga, Latvia  
*Guntis.Arnicans@lu.lv, Vineta.Arnican@lu.lv*

This paper describes the Sponsor-User-Programmer (SUP) model, which offers simplified and structured views of certain aspects of software requirements. The principles of the model are based on the idea that there are three major players or stakeholders in software development – the client or sponsor, the user, and the programmer or software developer. Each of these people has his or her own vision and documentation about the behavior and features of the software that is being developed. The SUP model brings together software requirements, classifying them in easily perceived classes according to which stakeholders agree with them and whether they are realized. The authors give an insight into how project managers, stakeholders, and testers can use the SUP model. The relation between the SUP model and testing processes is described.

**Keywords:** requirement engineering, testing, testing process improvement, software process improvement.

### 1 Introduction

There are many methodologies for software development which offer a detailed description of the development process as such, as well as the testing process and its role in the overall design process. The availability of these methodologies, however, does not in and of itself mean that they are put to practical use. On the other hand, “[...] meeting real customer needs and improving project estimates and visibility with the customer were to be key drivers for the software process improvement” [1] in industry. Moreover, software development processes involve such people as clients and users, who are often quite unfamiliar with IT technologies and design methodologies.

In recent years, people without specific knowledge have become involved in testing procedures more and more often because of the ever more acute lack of software testing specialists. Without education and training, these employees do not ensure high-quality testing, nor are they truly aware of their role in the process at large [2]. The same is true of clients and those who finance software development. They, too, require simplified and basic knowledge to oversee the situation at hand. Hence, “modeling conventions,

methodologies, and strategies all help to simplify requirement engineering techniques so that the techniques can be used successfully by typical practitioners” [3].

The first conceptual version of the SUP model has been established to make it easier for the many persons involved in software development to understand the situation. This model, to put it simply, offers a bird’s eye view of many important aspects of the quality and testing of software. The model divides the persons involved in software development into three groups – *Sponsors* (those who commission the software, own it, and finance the process), *Users* (the target audience for the software that is being designed), and *Programmers* (those who prepare the software – programmers, system analysts, and other IT specialists). The first letters of the names of those groups form the SUP model’s name.

In the context of the SUP model, we use the term *Product* as “any deliverables of life cycle – code, documents, diagrams, etc” and term *Project* as “any piece of work we may need to test – project, on-going maintenance, emergency fix, etc”, like in [4].

The goal of introducing the SUP model is to improve the testing process by improving the understanding of all involved persons in terms of the work that is being done.

There are at least two views on quality expressed in literature [5–7]. One way of considering the quality of software is to determine whether it is in line with the requirements that have been stated. Such requirements, it is believed, are documented in a written form. That is particularly true in outsourcing projects, because documents help to uphold the formal relations between the client and the service provider.

Users, however, can have a different view of quality. They want to know whether the software meets their expectations. Software is of quality, in other words, if it is convenient and easy to use. This approach is typical in those cases when software is designed for a wide range of users, and profits depend on whether users are prepared to support the software by buying it.

The SUP model involves both of the aforementioned principles as equally important, and only the user of the model decides whether to take either or both of those principles into account. The first view of quality is based on the fact that requirements vis-à-vis the software have been documented. Thus, the SUP model includes the “*Document View*” (SUP-D). The second principle is more difficult to quantify because the understanding of quality is based on the feelings, expectations, and visions of individuals regarding the software that is being designed. This can be called the “*Vision View*” (SUP-V). The model takes into account the fact that there can be differences between an individual’s vision of the product, on the one hand, and the written requirements which have been prepared by that individual, on the other hand.

Testing activities must begin as soon as possible in the lifecycle of software development, and they must iteratively continue throughout the process [8]. This applies to the planning stage as well as to all others, and testing helps to ensure that the work that is being done is in line with all relevant stakeholders. The SUP model makes it possible to review the process at any of its stages and to determine the issues that have been inadequately addressed, as well as the issue of whether the testing has been sufficiently extensive and adequate.

The remainder of this paper is structured as follows: section 2 describes the concept of the SUP model. Section 3 demonstrates interpretation of SUP regions in detail, section 4 outlines how to use the SUP model in testing process improvement. Finally, section 5 presents conclusions and describes directions for future work.

## 2 The Concept of the SUP Model

The SUP model, in essence, is a set of various views of software under development. The main components in the model are the stakeholders—Sponsor, User, and Programmer; let us call them *Actors*—their views on the *Product*, both in terms of the Vision View and the Document View, overlap and create common areas of viewpoints. Each existing or future behavior or feature of the Product will be in line with one of these areas. The areas can be of different levels of importance in terms of how necessary for the Actors are the requirements contained in each of them. Product quality is improved when steps are taken to reduce the number of requirements in less important areas and to enhance the scope of the important areas.

### 2.1 Origins of the SUP Model

The ideas behind the SUP model have been developed over the course of several years of investigation why software testing is often inadequate and incomplete, why resources are frequently wasted in the process, and why it occurs spontaneously and without any clear planning ever so often. We determined that the main problem was that design teams have mediocre or even poor knowledge about software testing. Software sponsors and users, of course, have even poorer knowledge. Each stakeholder has a different understanding of testing, and that can lead to disagreements.

The search was on for a model that all stakeholders in software development could understand, one which would offer understandable concepts that can be described in a single presentation or lecture. The initial model was largely based on the principle described in [9]—that software behavior can be discussed from the viewpoint of specification and/or of the actual software. It is the viewpoint of software developers to software testing. Testers need to look for differences between these two sets of information. Test cases are used to test various parts of the process, both in terms of the areas of viewpoints and in terms of issues which are outside those areas. Different testing methods test software behavior in different ways. Visual images (Venn diagrams) help readers understand better what the methods do and do not test.

### 2.2 Actors in the SUP Model and Their Views

The central role in the SUP model is played by several important groups of persons involved in software development and their views on the final Product.

#### 2.2.1 Actors and Their Classification

We have defined three important groups of stakeholders in the process of software development:

- 1) the Sponsor—the person or persons who commission the software, finance its design, and wish to gain material benefits from it;
- 2) the User—the person or persons who are the target audience of the software and will use it for work or personal purposes;
- 3) the Programmer—the person or persons who design the software and receive payment or other remuneration for their work.

There are cases in which a specific individual is in several of these groups, and in that case the individual is playing several roles simultaneously. That is good in the sense that the number of different viewpoints is smaller, but it is not good in the sense that the individual's viewpoint may be narrow and even erroneous.

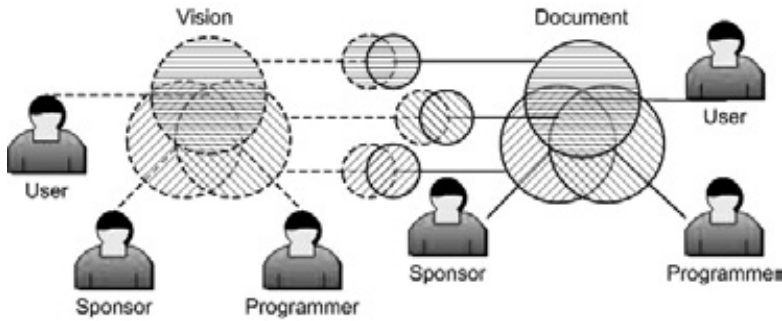





Fig. 1. Each Actor has a unique view (visions and documented requirements) of the Product. Here we see the interaction between the Sponsor, User, and Programmer.

Table 1 shows the graphic designations of the various states of software used in the diagrams of the SUP model.

Table 1

Graphic designations of the states of software in the SUP model

Shape	Meaning
Dashed area 	Ideas and visions about the software which are not written down
Solid line area 	Written requirements (specifications, software design, etc)
Filled area 	Requirements implemented in the software (the behavior and properties of the software, the properties of the source code, the properties of the documentation)

### 2.2.2 The Different Views of Stakeholders vis-à-vis the Product

Software and its aspects can also be reviewed at three levels:

- 1) the idea and vision as to what kind of software is needed and what its functions should be. We can regard this as the mental model of the software envisaged by an individual. Let us call this *Vision* (dashed area in the diagrams).
- 2) Visions, ideas, and requirements written down in formal documents, letters, diagrams, etc. These specify the requirements of the client, the requirements and complaints of the users, the proposed software design, etc. Let us call these *Documents* (solid line area in the diagrams).
- 3) The software which consists of the executable program, its code, and its support documentation – *Product* (filled area in the diagrams).

### 2.2.3 The Visions of the Stakeholders

Each stakeholder in software development will have his or her own ideas and visions about the behavior and properties of the software that is being designed. The closer those ideas are to the actual behavior and properties of the final Product, the more likely it is that the stakeholders will appreciate the level of its quality.

Each Actor has a different view on the necessary system behavior in most cases, and often these views are widely divergent. Figure 2 (a) shows the relationship among the various visions. Let us refer to the views of all involved Actors as the *SUP-Vision* (*SUP-V*). Each region is identified by a name – “S” for Sponsor, “U” for User, and “P” for Programmer or any combination of them. Each region here and in the rest of the paper represents graphically the appropriate set of unwritten (in *SUP-V*) or written (in *SUP-D*) requirements of software behavior and/or properties.

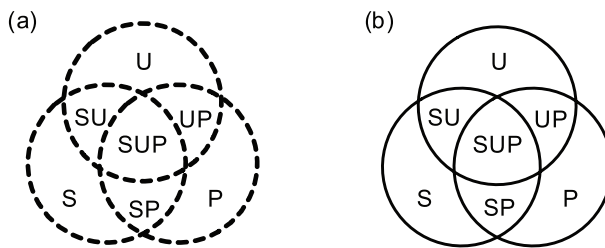


Fig. 2. The *SUP-Vision* View (a), the *SUP-Documents* View (b) and their segments

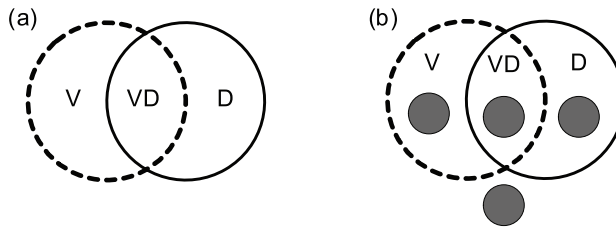
### 2.2.4 The Requirements Documented by Actors

The requirements of all Actors are identified and documented during the software development process. The developers come up not just with the source code but also, when needed, with additional specifications that are appropriate for the given situation – user handbooks, help systems, etc. We must remember that some of the documents prepared by the Sponsor and the User are also part of the final Product.

The documents can be classified in terms of their claims or requirements. Each person could look at each claim and declare whether it is or is not in line with the ideas of the relevant person vis-à-vis the software. By classifying all of the claims in the documents, we can define the view *SUP-Documents* (*SUP-D*, Figure 2 (b)). Each segment can be given an identifying name, as in the *SUP-Vision* above. It is assumed that the claim has been formally accepted by the person to whose range of interests the relevant segment belongs.

### 2.2.5 The Links between Visions and Documented Requirements

When systems are commissioned and designed, the ideas of each of the stakeholders in the process are documented. The User hands in written requirements, the Sponsor sets high-level requirements and specifications, the Programmer prepares the initial designs of the software, supplements specifications, develops the code, writes the user handbooks, etc. The fact is, however, that not all wishes are put in writing, and not all wishes are realistic. Figure 3 (a) shows this situation, V represents an area related to *SUP-Vision*, D represents an area related to *SUP-Documents*.



*Fig. 3.* The relationship between Vision and Documents and Product from the viewpoint of the User, Sponsor, or Programmer: (a) no requirements implemented; (b) some requirements from each region have already been implemented

Now let us review Figure 3 from the perspective of the User. We see that their Vision (the dashed area – V and VD) is different from the written requirements (the solid line area – VD and D). The area marked as VD is where the vision coincides with the requirements vis-à-vis the properties of the software. It is all but impossible, however, for users to put absolutely all of their desires into writing. The requirements which are not described are in the area V. There can also be mistakes in written requirements – ones that are not really in line with the User’s wishes and vision. These are in the area D. The filled areas in Figure 3 (b) represent requirements from each region that the Programmer has already implemented.

If we look at Figure 3 from the viewpoint of the Sponsor or Programmer, we see that the situation is analogous to that of the User, as described above.

In a perfect world, V and D would coincide. All the wishes of the User, Sponsor, and Producer would be written down, and that which is written down would be fully in line with that which is desired. Thus, all the wishes and ideas of the Actors would be included in the initial design, the documentation, and the later phases of the lifecycle of the software. Furthermore, everything would have been accomplished without any mistakes or misunderstandings. Of course, that never happens in real life because of the significant cost of software development and because of the fact that the various Actors inevitably have at least a few contradictory wishes.

### 2.2.6 The Interaction between the Views of Actors

As we saw in Figure 2, some of the ideas or visions are common to all Actors – what the User and Sponsor want the Programmer is ready to provide (SUP). Some ideas are common to the Sponsor and User (SU), some are common to the User and Programmer (UP), and some are common to the Sponsor and Programmer (SP). Each stakeholder usually has some idea about the software that is unknown to the other stakeholders or to which the others do not agree (“S” for Sponsors, “U” for Users, and “P” for Programmers in Figure 2). The situation with documented requirements is similar.

## 2.3 The Product and Its Relationship with Views

We have so far focused on the vision and written requirements related to the Product. The job is to produce a Product that is as close as possible to the Vision of all relevant Actors. Some of these Visions will be documented as requirements. Only

some will end up in the final software source code (in Figure 3 (b), the filled areas represent the produced software behavior and properties that are in line with the requirements in the relevant segments). The sad fact is that different people interpret requirements in different ways, which results in mistakes. This means that part of the Product will satisfy the Vision, another part will satisfy the Document (documented requirements) and the third part will satisfy both (the filled area in the segments in Figure 3). The Product, moreover, can have behavior and properties which are not in line with the desires or requirements of the Actors (the filled area outside the segments).

A few interpretations, for instance, from the perspective of the User:

- in segment VD, where visions and documented requirements coincide, we see that there are requirements which have already been implemented. They are in the filled area, and that means that not all requirements have yet been implemented, because the entire segment has not been filled in.
- If the software behavior and properties are in line with items from segment V, that means the Programmer has taken into account the User's wishes that have not been set out in writing but that are nevertheless necessary and in line with needs.
- If the behavior and properties are in line with items from segment D, that means the Programmer has implemented mistaken requirements of the User (i.e., the written document does not truly reflect the wishes of the User).
- Software code that is outside the segments V, VD, and D refers to everything else that has been done – items about which the user has neither visions nor requirements. These may be specific requirements from the Sponsor, items which simply make it easier to maintain and test the software, as well as functions nobody needs.

The (a) and (b) parts of Figure 4 together depict the structure of the SUP model, which is its basic framework for testing needs. One conclusion usually drawn by those who study the SUP model is that testing must be very diverse to check many different issues (Figure 4 (b) alone has 16 different segments – 7 segments, for instance, U, SU, UP, and 8 filled circles that represent requirements implemented in software according to each segment).

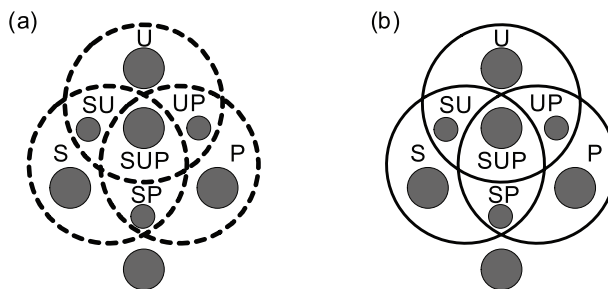


Fig. 4. SUP views during software development – (a) visions about the software (the SUP-V model) and the resulting software; (b) documented requirements for the software (the SUP-D model) and the resulting software

## 2.4 Regions of Views

### 2.4.1 Regions: the Cornerstone to Classify and Evaluate Situations

Because of the varying number of Actors, there can be various combinations in terms of situations that can be represented in the relevant region. It is not difficult to find the real-life interpretation of each region. Let us consider some of these. Various methods can be used to identify and process any situation.

Not all situations are equal in terms of necessity and utility. Furthermore, interpretation of the situation will differ depending on the current phase of the project's lifecycle. Each region in the SUP model can be weighted so as to emphasize the fact that not all regions are equal at any given time.

Let us consider an instance of the project deadline drawing near. The greatest weight will be in the filled-in field of the SUP region. That means that the views of the Sponsor, the User, and the Programmer coincide regarding the behavior and properties of the software, and the relevant requirements have been implemented in the Product. To us, this is a desirable situation.

Next, we could consider the filled areas in the SP, UP, and SU regions. These refer to those parts of software requirements which lack the support of just one of the Actors. Relationships among these regions depend on which Actor is dominant.

Unpleasant situations which are less important are represented in the open areas in the regions – the desired or required behavior has not been ensured. The worst of this situation is the open area in the SUP region at the end of the project, because there unanimity on requirements is achieved, but those requirements have not been implemented in an executable code.

A less clear situation exists with software which no one really needs – software which has perhaps been created by mistake (the filled-in field outside the regions). The open area outside the regions represents unknown future potential, both in a positive and a negative sense.

When a specific project is developed, the weighting of various regions can occur on the basis of specific principles. Within a single company, however, a weighting principle that is valid for more than one project can be developed. This is done while taking into account the company's specifics and culture.

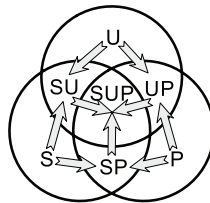


Fig. 5. Paths from regions with smaller weight to regions with greater weight

### 2.4.2 Strategies for Quality Improvements from the Perspective of Regions

Strategies for quality improvement with the help of the regions can be described in a simplified way:

- 1) identify requirements of the Product in each SUP model's region;



- 2) set a goal in terms of the target region in which we wish to see each identified requirement. The goal may be to throw out the unnecessary requirement, too;
- 3) choose the pathway from the initial region to the target region, moving to neighboring regions which are of greater weight (if there are several pathways, we choose the optimal one);
- 4) take steps to achieve our goals.

The first of the steps is primarily based on various testing methods. The fourth step can be taken by changing the Product appropriately or by using explanations and arguments to change the views of other Actors regarding the Product.

## 2.5 The Dynamics of the SUP Model

In working with the SUP model, we must remember that its components change their status over the course of time. That is primarily because people change their minds about things. They like things which they used to dislike and vice-versa. If Document View has changed, the documents become erroneous. Those who maintain the software can make changes which are no longer in line with the vision of the documentation of an Actor.

Depending on the segment that is affected, changes can be radical or minimal; positive or negative. The SUP model can help judge how important the changes are and decide whether they are, generally speaking, desirable or undesirable.

## 3 Interpretation of SUP Regions

One of the main goals of the SUP model is to inform Actors about typical situations that are described in the regions so that the software testing process can be aimed at identifying those situations; furthermore, steps can be taken to change them in the desired way.

Let us look at the situation in Figure 4 (b) (relations between SUP-D and the Product).

- **The filled-in part of the SUP region** is the ideal situation in terms of this view because it represents a set of specifications about which the Sponsor, User, and Programmer have reached a written agreement. The Product fully corresponds to those specifications. We want this area to be as large as possible. In terms of testing, it would be enough if only a couple of tests showed correspondence to this section. Additional tests ensure greater psychological security, but they really are a waste of resources. Examination of sample tests will not change the Product quality. The tests that are available in this segment are very useful, however, to explain the specifications more precisely, to conduct testing during the software development process, and to offer examples in the user documentation that is used.
- **The open area in the SUP region** represents the issues which all stakeholders have agreed upon but which have not been implemented in the code of the Product for one reason or another. One must carefully consider how the relevant properties can be ensured in the Product and what resources are needed for that purpose. As the design process draws to a close, testing must focus primarily on this region, because the existence of such region makes it clear that the Programmer has failed in one

way or another. We must remember, however, that from the perspective of SUP-V, if the relevant behavior is in a different region, the specifications must be changed. It is quite possible that there is a fundamental reason for such a situation, and that reason must be identified.

- **The filled-in area in the SU region** represents things which Users wanted and the Sponsor specified and which the Programmer did in opposition to their own official view. If the Programmer can officially accept that, the requirements of functionality move to the filled-in area of the SUP region (this is required because otherwise the Programmer can just remove functionality from the Product).
- **The open area in the SU region** represents things which Users want and have specified, but which the Programmer cannot promise and perhaps has no intention to do. If compromise can be found as the project develops, the situation might move from this region to the SUP region.
- **The UP region** – the User has written down requirements which the Programmer is prepared to accept (the open area) or has already implemented (filled-in area), but which the Sponsor is not going to finance or which are not included in project specifications for any other reason. This situation often occurs in in-house projects, when the Programmer communicates directly with the User, but specifications are never really put together because of lack of time or other reasons. What happens next depends on whether the relevant request has already been satisfied. The User and/or Programmer can attempt to convince the Sponsor to change the specifications. It is also possible that everyone is satisfied with the situation, even though it has not been put into the specification – resources have been saved! Another possibility is that a key goal of the Sponsor is not pursued precisely. For instance, the Sponsor may want to limit user access to data, while the User wants to access all data (the latter is simpler to implement for the Programmer). In that case, the developed code may have to be revised.
- **The SP region** represents things which have been listed in the specification, implemented in the software, or are to be implemented by the Programmer, but which the User does not need. The reason may be that the User is simply not interested in them. This applies to things such as the security system, testware to make testing easier, handling of mistakes, or establishment of an audit trail. It may be, however, that because of incompetence, functions that are bothersome or even hazardous for the User are requested and installed. The User and the Sponsor alike need to be convinced to reach an agreement on what to do. The Sponsor, of course, is free to spend his/her money as he/she sees fit; nonetheless, specifications can be mistaken.
- **The S region** represents things that are listed in specifications, are not demanded by the User, and which the Programmer does not intend to implement. These situations can move to the SP region as the Programmer continues their work, but it is also possible that the requirements which are found here will never be pursued because the User categorically objects to them. The response here is similar to that in the SP region.
- **The U region** represents things which the User wants but the Sponsor is, for one reason or another, unwilling to put in the specifications for the Programmer. The Users can communicate with the Sponsor to ensure that the requirements are placed in the specifications (in which case the situation moves to the SU region), or they can

deal directly with the Programmer (PU), reaching an agreement with the Sponsor afterwards. Sadly, this is quite common in real life, because the Users are often brought into the development process very lately or not at all.

- **The P region** represents behavior or properties of software that are denoted in the software development or in the documents which the Programmer has put together for their needs, but which have been requested neither by the User nor by the Sponsor in their specifications. Perhaps such functionality is unnecessary, or it is necessary but neither the Sponsor nor the User has thought of it, therefore it is not in the specification. This often applies to functionality the technical personnel that run the system need. This can concern things such as the LOG trail to find out what work or calculations are being done. Specifications usually do not include handling of low-level errors.

If sufficient real life examples are given when potential users of the SUP model are taught about the SUP model, they quickly learn to identify situations and to see what should be done in response to them.

## 4 Improvement of the Testing Process

Requirements are at heart of each software development project. Testers have to do both – validate compliance of requirements to stakeholders' claims, and verify correctness of implementation of requirements through all stages of software development.

### 4.1 Support of the Testing Process Management

Exploitation of the SUP model can in and of itself improve the testing process because structured knowledge about all possible situations make it possible to view testing in a different way and to restructure the relevant activities. The model defines directions to pursue, but it does not address testing in detail.

The SUP model can be a tool that helps start testing early, do it thoroughly and differently, guides planning of testing through the whole lifecycle of software development. All these issues are essential to quality testing [8]. In such a way SUP model supports iterative reassessment of the quality of testing itself in order to improve testing processes [4, 10, 11].

Testing allows project managers to know facts about software – what kinds of faults were found in the software, how many faults were found and where in the software they were found. Quality testing also shows what kinds of faults have been sought but not found. This is an iterative process. Each time the project is reviewed there must be consideration of whether any changes in the project have been forgotten. Thus, it is possible to reassess the entire project situation and to make informed decisions on improving the relevant processes.

The SUP model makes it possible to improve software quality by highlighting project weaknesses, such as requirements which the Programmer has followed by request of the User but which are not found in the Sponsor's specifications.

The SUP method makes it possible to use resources more rationally – not at the end of the software development process, which is when testing usually takes place. It makes it possible to learn early on that the requirements of the Sponsor (the S region) are not

in line with Users' wishes, so the Users will not accept them (in which case the situation is in the SP region). The model makes it possible to monitor whether the Sponsor has agreed upon all specifications that will affect the User before the Programmer starts the work (the desired movement is from the S region to the SU region and then to the SUP region).

## 4.2 Seeking Problems in a Project Using SUP Views

At the beginning of a project, when requirements are first being developed and the goals of the system are being designed (SUP-V view), all wishes are going to be in the S, U, and P regions. Testers can work with requirements, for instance, from the U region. They can do usability testing, reviews of these requirements in order to help the Users:

- specify their wishes;
- see which requirements are contradictory;
- prepare to advocate their requirements in the eyes of the Sponsor.

Later the SUP-D view of the model is used in testing to ensure a proper view of the project as a whole:

- the requirements on which the Actors have and have not reached an agreement;
- which requirements have been and have not been fulfilled in the software;
- recollection that there may be additional functionality that is not specifically requested.

Views of SUP model give to testers a clear sight on requirements.

- What issues is each Actor thinking about (regions S, U, P)?
- Which requirements are going to be discussed in the future (regions SU, SP, UP)?
- Which requirements are implemented in each region? Which are not? Why?
- Is there functionality in the software that is not included in the requirements? Is it really redundant or not (for instance, if requirements from UP region are implemented – issues that the User needs, the Programmer agrees on with the User, but the Sponsor is not ready to include in the official project specification)?
- Are the documented requirements truly those of the User, Sponsor and Programmer?
- Are there contradictions among the various requirements?
- Is the set of requirements complete? For instance, are there requirements with which one of the Actors is not particularly familiar, perhaps requirements in which certain Actors have no interest at all? Usability requirements which are not directly related to business requirements are often, and sadly, examples of this kind of situation. For instance, sometimes specifications state the maximum amount of time that can pass before the system responds to a user request. Seldom, however, is there an indication of the minimum font sizes on the screen – something that is important to users, given that different people have different ability of vision.

Simultaneously with the job of harmonization the Users, Sponsors, and Programmers do, testers can start to prepare use cases, scenarios, test specifications for future testing, feeling the real needs of the User and Sponsor.

### 4.3 Choice and Use of Testing Methods

While the requirements have not yet been implemented, testers use only static methods in their work – inspections, reviews, prototype usability testing on paper.

The overall lexicon and collection of requirements of the project is constantly supplemented and expanded in collaboration with other project developers.

Testers also prepare testing plans and user scenarios for those requirements which are already in the SUP region. They give thought to the dynamic testing that will be necessary in the future.

As soon as parts of the Product are implemented, testers continue the work that they have done before as well as start to inspect whether the requirements and the executable code are in line with one another.

- Have the requirements been implemented correctly?
- Have any requirements on which the Actors have not yet agreed been implemented?
- Does the code include unnecessary functionality which no one has requested?

During this period, testers use traditional dynamic test methods to make sure that all requirements have been realized implemented as intended. Implementation of unexpected requirements or of completely unnecessary functionality is identified by structural testing methods (the white box methods), exploratory testing, *ad hoc* testing, and code reviews.

Typical black box testing methods, which are based only on the written requirements of the Sponsor, are operational from the SUP-D view and in relation to all regions which involve the letter S. They have little to do with interests of the User, and they cannot be used to identify unnecessary programming in the region P or outside all of the regions.

Typical white box testing methods, by contrast, use the source code, but they cannot in and of themselves check the regions S, SU, and U because essentially they are valid for all regions which involve the letter P.

Static testing methods inspect the source code while simultaneously reviewing requirement documents. Testing occurs in all regions, but there are two key shortcomings here: testers must be highly qualified, and it is not possible to identify specific errors that affect the behavior of the software (e.g., testing the speed of reaction of the system).

Acceptance testing, which is very popular and often greatly relied upon, basically operates only in the SUP region, and that does not represent adequate testing.

## 5 Conclusions and Future Work

The SUP model described in this paper was established gradually as a means for improving the testing process in order to help address certain problems that exist in practice. The most important aspect of this model is that it offers a bird's eye view of software and its functionality from the perspective of the various Actors. The model can be used by IT specialists as well as by various businesspeople whose knowledge on IT and computers can be skimpy or incomplete.

The SUP model classifies the many situations that can emerge from the visions and written specifications of Actors and from the resulting software. When such situations are explained with understandable examples, it is easier to achieve better understanding

of the testing process among the many persons involved in a software development project. Then it is clear whether one of the Actors is being ignored, e.g., whether all potential users are or are not brought into the discussion to an adequate degree. It is also possible to tell whether testing is not too unilateral or narrowly drawn.

The User and Sponsor often do not have enough time to inspect the whole process, so a narrower model can be proposed to them – just the SUP-V or SUP-D view, with explanations of the relevant regions.

The SUP model can be used as an additional resource for better organization of software development processes throughout the lifecycle of the project. The model supports the launch of testing activities at the stage of initial visions, helping to plan the project and analyze possible risks. In later phases, the model can be used to develop a strategy for software testing, plan the testing, review and improve the plans, and select the necessary testing methods and techniques.

The SUP model makes possible an extensive testing of software in order to make sure that there is nothing distinctly unacceptable about the software development process or the Product itself.

The SUP model is used as an additional resource in real projects. Students who are trained as testers learn about the model at university and elsewhere. Up to now, the model has been used informally and intuitively. Further research is needed to formalize the SUP model and to define a more precise methodology for its use. Several algorithms or scenarios should be prepared for the use of the model, and guidelines for their use should also be created. Particular attention could be given to the issue of how to determine the existing condition of a project best – by identifying the problematic regions in the SUP model, by highlighting critical problems or weaknesses, and by clearly understanding what must be done to improve the situation. This can be achieved by analyzing all project-related activities – meetings, the involved people, the drafted documents, etc. Documents can be inspected, and everyone involved in the project can be surveyed.

## Acknowledgements

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

## References

1. O'Hara F. (2000) European Experiences with Software Process Improvement. Proceedings of the 22nd International Conference on Software Engineering, Limerick, 635–640.
2. Arnican V. (2007) Use of Non-IT Testers in Software Development. In: Münch J., Abrahamsson P. (eds.) *Product Focused Software Process Improvement. Lecture Notes in Computer Science*, Vol. 4589. Berlin-Heidelberg: Springer-Verlag, 175–187.
3. Cheng B. H. C., Atlee J. M. (2007) Research Directions in Requirements Engineering. *Future of Software Engineering* (FOSE '07), IEEE Computer Society, 285–303.
4. Veenendaal E. (2002) *The Testing Practitioner*. UTN Publishers, Den Bosch.
5. Lewis W. E. (2005) *Software Testing and Continuous Quality Improvement*. 2<sup>nd</sup> edn. Boca Raton, London, New York, Washington, D.C.: Auerbach Publications.
6. Conradi R., Fuggetta A. Improving Software Process Improvement. *IEEE Software*, Vol. 17, No. 4, IEEE Computer Society, July/Aug. 2000, 76–78.

7. Van Solingen R. (2000) *Product Focused Software Process Improvement. SPI in the Embedded Software Domain*. Eindhoven University of Technology, Eindhoven.
8. Perry W. E. (2000) *Effective Methods for Software Testing*. 2nd ed. New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc.
9. Jorgensen P. C. (2002) *Software Testing: A Craftman's Approach*. Boca Raton, London, New York, Washington, D.C.: CRC Press.
10. Palyagar B., Moisiadis F. (2006) Validating Requirements Engineering Process Improvements – a Case Study. First International Workshop on Requirements Engineering Visualization (REV'06 – RE'06 Workshop), 9.
11. Huo M., Zhang H., Jeffery R. (2006) An Exploratory Study of Process Enactment as Input to Software Process Improvement. Proceedings of the 2006 International Workshop on Software Quality. Shanghai, 39–44.