Meaning as Use: Handling Requirements Conflicts and Inconsistencies

Andrés Silva

Facultad de Informática
Universidad Politécnica de Madrid
asilva@fi.upm.es

Abstract

In Viewpoint-based Requirements Engineering (VBRE) information is elicited and documented from multiple viewpoints. At some point in a VBRE process, the contents of different viewpoints should be compared. But in Requirements Engineering (RE) we not only deal with "requirements". If we take into account the multiple uses of the information managed in a requirements process it appears reasonable that, before checking for discrepancies, the role of each element to be compared needs to be cleared: requirements, for example, should not be compared against descriptions. Also, interesting relations between discrepancies could be found: a conflict between requirements, for example, could be due to inconsistent descriptions. In this paper a VBRE process is proposed based on a previous categorization of the contents of each perspective according to their use. The framework proposed includes detection and classification of discrepancies and solution generation.

Keywords: Viewpoints, requirements conflict, requirements inconsistencies.

1 Introduction

There is a multiplicity of stakeholders that take part in every requirements process and this will inevitably lead to discrepancies. Viewpoint-based Requirements Engineering (VBRE) recognizes this fact, but tries to obtain benefits from it. In VBRE discrepancies are not seem as undesirable: instead, adequately managed, discrepancies between stakeholders can be used as a means to improve the elicitation of requirements and to improve other aspects of software development [NER00]. But a discrepancy management process could be improved by taking into account that in Requirements Engineering we deal with informations of different nature [ZJ97, Par95] like descriptions, goals or specifications. Discrepancy management should be done after clarifying which of those categories the potentially discrepant statements belong to.

In this paper the term discrepancy will embrace conflicts and inconsistencies. A conflict is the particular case when there is some kind of clash between goals. An inconsistency happens when there is a clash between descriptions. Conflicts and inconsistencies could lead to clashing specifications. We will introduce a process for detection, classification and resolution of discrepancies, based on a previous categorization of the contents of each viewpoint. One of the goals pursued by this process is the ability to deal with viewpoints expressed in formalisms ranging from natural language to formal specifications. This would free the process from the idiosincracies of each particular modelling language, at the cost of being more definite.

1.1 Different uses of a statement

The consideration of a multiplicity of categories of information according to their use is not new. Ludwig Wittgenstein pointed out in a famous aphorism of his "Philosophical Investigations" [Wit53] that a picture, depending on its use, could not only be descriptive:

> (22): Imagine a picture representing a boxer in a particular stance. Now, this picture can be used to tell someone how he should stand, should hold himself; or how he should not hold himself; or how a particular man did stand in such-and-such a place; and so on. One might (using the language of chemistry) call this picture a proposition-radical.

Wittgenstein calls the picture a proposition-radical because it needs to be complemented with its use (descriptive, imperative, etc.) to be a real proposition.

In Requirements Engineering something similar happens. Requirements are collected and modelled using some formalism, from natural language to formal specification languages but, very often, the use of a sentence is not clear. For example, during the development of a system to control a lift [Jac95a], we could find two statements like these:

- "When the lift is stationary at a floor its doors *are* open"
- "When the lift is stationary at a floor its doors *are never* open"

These sentences are nothing if we ignore their use. Are they descriptions? (i.e. statements that pretend to describe a piece of reality, and whose truth value is unrelated to the existence of any software system); are they requirements? (i.e. statements that express goals, whose truth value is, of course, related to the existence of a future software controller). If they are requirements, how could a software controller enforce them? But the important question for VBRE is the following: how can we compare one sentence against the other before their use is clear? For example, both expresions could be seen as descriptions of actual facts (some lifts could use a mechanical device to actually behave in one, or the other, way). In that case, one of them must be wrong, and the way to elucidate which of them is wrong is by means of refutation, or checking them against the facts. On another side, both expresions could be goal statements. In this case, we can not refutate this sentences against reality but we could start a negotiation process to solve the conflict. In the particular case in which one sentence is descriptive and the other is a goal, we should not compare one against the other, as this comparison makes no sense.

1.2 Knowledge, Specification and Requirements

In Requirements Engineering we deal with not only one category of statements, but with different categories at the same time. The KSR model, proposed by M. A. Jackson [Jac95b, ZJ97] takes this into account, distinguishing between three categories contained in three sets K, S and R:

K: Contains descriptive sentences. This sentences are true with independence of what the system will or will not do.

R: Contains the requirements. Requirements, or goals, describe the desired effects of building the system and deploying it in a particular environment.

S: Describes the external behaviour of the machine, in other words, describes the interface between the machine and its enviroment. The task of defining the contents of S is a task full of design decisions. Only shared phenomena can be part of the sentences contained in S.

The relation that should hold between the three sets is: $K, S \models R$. So, given the domain K and the specification S, the requirements R will be carried out, i.e. they will become true in the environment.

Before checking for discrepancies between viewpoints, if we classify their contents into the sets K, S and R, we could make comparisons between each set and relate the discrepancies found in one category to the discrepancies in another category. We could, for example, relate a discrepancy between elements of K to another one between elements of S.

The remainder of this paper is organized as follows: section 2 introduces the concepts of overlaps and discrepancies as used in the paper; section 3 shows the proposed classification of discrepancies; section 4 shows the resolution tasks and its relation with the proposed classification of discrepancies; section 5 shows the phases of the proposed process for discrepancy detection, classification and resolution; finally, section 6 provides some results and conclusions.

2 Overlaps and discrepancies

Discrepancies can only emerge between statements about the same phenomena. From this point of view, two sentences like "all human are mortal" and "this book is a novel" can never be discrepant. If we consider that predicates denote phenomena and well formed formulae (wffs) of the predicate calculus denote statements over phenomena, we can say that a set of wffs can be discrepant only when they share some predicates. These overlaps [SFT99] between formulae are a precondition for an inconsistency, or a conflict.

2.1 Overlap criteria

In this work we propose the use of user-defined overlap criteria, *oc*. The ability to define these criteria incorporates some degree of flexibility that is needed in RE. For example, if p and q are given as:

$$p \mathrel{\widehat{=}} \forall x(book(x) \wedge edited(x, London) \rightarrow on\_catalog(x))$$
$$q \mathrel{\widehat{=}} \forall x(book(x) \rightarrow \exists y(writer(y) \wedge author(y, x)))$$

we can say that p and q overlap with respect to an overlap criterion like:

$oc \mathrel{\widehat{=}}$ Two wffs overlap if they share at least one predicate

but p and q would not overlap with the criterion:

$oc \mathrel{\widehat{=}}$ Two wffs overlap if they share at least two predicates

The overlap criteria can be extended from wffs to sets of wffs. Also, different criteria could be used for different formalisms. For example, requirements are commonly written in natural language. For natural language an overlap criterion could be as informal as:

$oc \mathrel{\widehat{=}}$ Two sentences overlap if both make reference to the same issue

The availability of tools that automate, or facilitate, the overlap detection process, can be used to define an *oc*. Also, even for the same requirements formalism, different criteria could be used depending on the state of the requirements process: In early phases it makes less sense to use a very restricting criterion, but in later phases, the criterion should be constrained.

2.2 Discrepancy criteria

Every representation or modelling formalism is based on sintactical and semantical rules that users should follow. Those rules define what a correct representation is, against an incorrect representation. As a consequence, the rules could be used to detect discrepancies between alternative models or alternative descriptions. For example, two Data Flow Diagramas (DFDs), corresponding to two different viewpoints, are discrepant if different input or ouput flows are related to the same process. So, for each particular notation, or combination of notations, we can define discrepancy criteria (*dc*) that will establish what is, and what is not, a discrepancy. A chosen criterion could not only depend on the notation but on other circumstances, like the state of the requirements process, or the availability of tools that facilitate the task.

It makes sense to speak only of discrepancies when referring to a set of wffs. For example, let $P = \{p, q\}$, where:

$$p \mathrel{\widehat{=}} \forall x(book(x) \wedge edited(x, London) \rightarrow in\_catalog(x))$$
$$q \mathrel{\widehat{=}} \forall x(book(x) \wedge edited(x, London) \rightarrow \neg in\_catalog(x))$$

In this set P, from a predicate calculus point of view, there is no discrepancy between p and q, as they are not inconsistent: any situation where no book is edited in London is a model for both formulae. But it would be correct to say that P is discrepant with respect to another criterion given as:

> $dc \hat{=}$ Two formulae are discrepant if their consequentes are inconsistent when their antecedents are identical.

When requirements are expressed in natural language, a discrepancy criterion could be someting as informal as:

> $dc \hat{=}$ Two sentences are discrepant when the reader perceives that both can not be true at the same time

For requirements expressed in different formalisms some criteria could be defined taking into account required correspondences between different models (for example, the required correspondence between elements of a DFD and an entity-relationship diagram).

3 Classification of discrepancies

The main criteria we propose for classifying discrepancies is based on the sets K, S and R that are affected in each particular case. The symbol ($'$) will be used to denote that a discrepancy exists between some elements of one of those sets. For example, a discrepancy between viewpoints that affects sets K and R is a discrepancy $K'SR'$. This classificaion has two advantages: (i) it is based on the previous classification of the contents of each viewpoint and (ii) it directs the generation of solutions, as each resolution task strongly depends on the sets affected: it is not the same, for example, to achieve a common consensus about the environment than to achieve an agreement about the goals of the system.

The eight types of discrepancy are shown in table 1. In this table, $K'SR'$ denotes that there is a relationship between the elements that are discrepant in K and the elements that are discrepant in R, so solving one could help to solve the other. But, if there is a discrepancy in K and another in R with no overlap relation between them, it would be just a $K'SR$ discrepancy, on one side, and a KSR' discrepancy, on the other. Table 1 has been divided in two parts. First part shows the mixed discrepancies, so called because they affect more than one of the three K, S, R sets. Second part shows the trivial discrepancies. In this part, the case of total agreement KSR has been included just for the sake of completeness, even when properly speaking it is not a discrepancy. The three cases where only one set is affected ($K'SR$, $KS'R$ and KSR') denote that the origin of the discrepancy is not in a conflict or an inconsistency, but something more trivial as a mistake (like a typo). This happens because it is impossible to disagree in only one set and still verify $K, S \models R$. In our work we are mainly interested in mixed discrepancies.

Mixed discrepancies		Trivial discrepancies	
K'S'R	Related discrepancies in K and S	KSR	No Discrepancy
K'SR'	Related discrepancies in K and R	K'SR	Discrepancy only in K
KS'R'	Related discrepancies in S and R	KS'R	Discrepancy only in S
K'S'R'	Total Discrepancy	KSR'	Discrepancy only in R

Table 1: Types of discrepancies.

4 Resolution tasks

In our proposal, the resolution process of discrepancies can be divided into lower level tasks, so each particular type of discrepancy can be solved by properly combining different tasks. Following, a brief description of the tasks is shown and table 2 shows the particular tasks that should be applied to each type of discrepancy.

Refutation. This task compares discrepant sentences against reality, to see which of them are true. This task is applied when there are discrepant elements in K, and it makes no sense to apply it to R or S.

Transference. This task eliminates wrong knowledge K when it is recognised that is wrong. It also substitutes this wrong knowledge by newly (acknowledged) correct one.

Checking. This task determines if a viewpoint is internally consistent. In particular, this task should be enacted after a transference has been done, as new knowledge could lead to internal inconsistencies.

Negotiation. This task is aimed towards an agreement between discrepang goals of the system. For this reason, it is applied only when there are conflicts between different viewpoints.

Acceptance. In this task one, or both, viewpoints, after reaching an agreement with respect to the requirements, modify their requirements accordingly.

Adaptation. When a discrepancy between requirements is not due to different, conflicting goals, but is due to a disagreement between different perceptions of the environment, this task should be enacted. This task performs a modification of the requirements to adapt to the new situation.

Redesign. In many situations after descriptions of external phenomena change, or when requirements change, this task should modify the external description of the system (S) to adapt to the new situation.

Error fix. Sometimes what appears to be a discrepancy between viewpoints is not so, because its cause is just a minor error or mistake. Typing errors, syntax errors and transcription errors are, of course, possible, so this is the task aimed towards correcting them.

Table 2 shows that for each particular type of discrepancy, different tasks should be used. The rows in this table show the tasks and the columns show the different kinds of discrepancy that could happen between two viewpoints. For each type of discrepancy the table should be read from top to bottom. A dot (•) in a cell indicates that the task is needed to solve that particular kind of discrepancy. The resolution tasks for the discrepancies that only affect to one category of K, S or R have been ommitted from the table, as its resolution task consists merely on an error (mistake) fix.

	K'S'R'	K'S'R	K'SR'	KS'R'
Refutation	•	•	•	
Transference	•	•	•	
Checking	•	•	•	
Adaptation	•		•	
Negotiation				•
Acceptance				•
Redesign	•	•		•

Table 2: Resolution tasks according to the type of discrepancy

5 A Process for discrepancy handling

For these ideas about classification and resolution of discrepancies to became useful in a VBRE process, we have devised a process that assembles all the concepts seen so far. The process we propose for dealing with two discrepant viewpoints is represented in figure 1. In this figure VP_i,VP_j represent two viewpoints, and VP'_i,VP'_j represent the same viewpoints after some discrepancies have been solved. The three phases of the process are: Phase 0 (Preliminary), Phase I (Detection/Classification) and Phase II (Resolution). Figure 1 shows that phase II could be done as many times as needed until an adequate number of discrepancies has been solved. Also, the modified viewpoints that result after phase II could enter phase I again, as the process of solving discrepancies could introduce new ones. The remainder of this section is devoted to a detailed explanation of these phases.

5.1 Phase 0

In this preparatory phase, information elicited from two stakeholders is collected in viewpoints P_i and P_j and structured according to the categories K, S and R for each viewpoint. Each viewpoint should be internally consistent, verifying $K, S \vdash R$. This partition of the knowledge internal to each viewpoint is needed because stakeholders do not only express their requirements: they could also contribute with some domain knowledge that could lead to disagreements with other stakeholders. Also, in this phase, overlap and discrepancy criteria are defined.

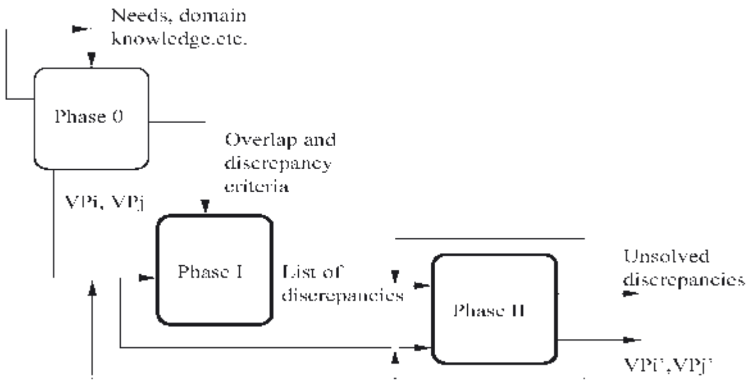

Figure 1: Main phases of the process

5.2 Phase I

This phase, using the criteria and the viewpoints structured in Phase 0, produces a list of all discrepancies between them, including the elements affected and its classification according to the criteria shown in section 3. Following, a simplified graphical description of Phase I is shown by an example of a $K'S'R'$ type of discrepancy. A formal, detailed description of this phase can be found in [SM01].

Let us suppose two viewpoints (A and B) whose contents have been structured in K, S, R, so $A = K_a \cup S_a \cup R_a$ with $K_a, S_a \models R_a$; and $B = K_b \cup S_b \cup R_b$ with $K_b, S_b \models R_b$. Let us suppose that S_a and S_b overlap. This situation is shown in figure 2. There could be a discrepancy between S_a and S_b, as they both overlap. This is shown in figure 3. This graphical representation abstracts away from the particular language used and the particular criterion of discrepancy (cd) used.

Once the minimal set of discrepant elements in S has been identified, we should now try to see if this discrepancy in S is, in some way, related to a possible discrepancy in K. We do this by obtaining the elements of $K_a \cup K_b$ that overlap with the discrepancy found in S, as shown in figure 4. Inside this subset of K, we find another discrepancy, shown as a gridded set in figure 5. Now we need to check if this discrepancy found in K is related to a discrepancy in R or not. First, we find the elements of $R_a \cup R_b$ that overlap with the discrepancy in K, as shown in figure 6. Secondly, we find a discrepancy affecting those elements, shown in figure 7.

Now we have identified the type of discrepancy between viewpoints A and B (a K'S'R' discrepancy, in this particular case) and we have knowledge of the discrepant sets that affect each internal category of A and B. In this example we started with a discrepancy in S. In the case that another discrepancy in S exists, the process would be repeated again. When no discrepancies in S are left, the process will try to find discrepancis in K and R, following an analogous procedure. In summary, the output of the process is a list of all the discrepancies between A and B, properly classified and including the discrepant elements that have been found (that is, the gridded subsets in our figures).

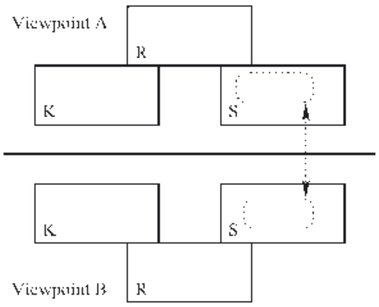

Figure 2: Overlaps in S

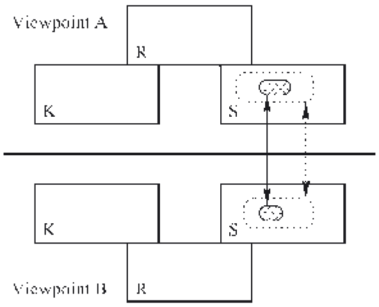

Figure 3: Overlap and discrepancy in S

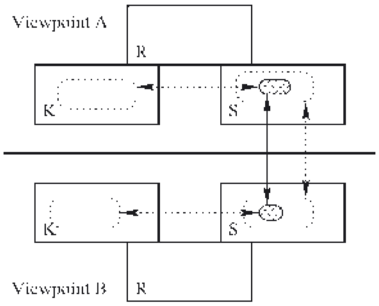

Figure 4: Finding the elements of K overlapped with the discrepancy found in S

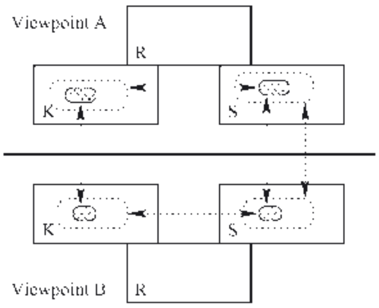

Figure 5: Finding the discrepancy in K

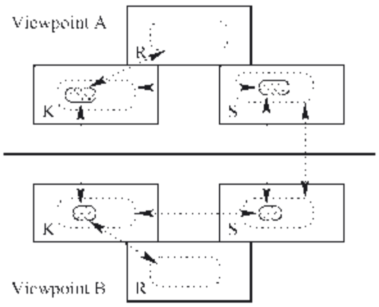

Figure 6: Elements in R overlapped with the discrepancy found in K

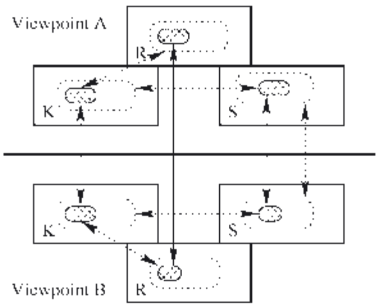

Figure 7: Discrepant elements in R, related to the discrepant ones in K and the ones in S

5.3 Phase II

Taking as input the original viewpoints P_i and P_j and the list of discrepancies, this phase attemtps to solve them. Of course this phase does not pretend to provide a complete automatization of the process: its main focus is on assisting the resolution task by generating solutions according to each discrepancy type. The output of this phase consists on the modified viewpoints and a list of unresolved discrepancies, as not all discrepancies must be solved immediately in a VBRE process [NKF94]. The resolution process proposed is based on the vocabulary of tasks introduced in section 4 and follows the scheme of figure 2.

5.4 Example of K'S'R

With the aim of showing, in a nutshell, how the process works, a small example is provided in this section. Let us suppose that two employees, A and B, have different (discrepant) perceptions about the behavior of a particular sensor, but they agree with the requirements. They both desire a software system that should be able to empty a boiler when the temperature reaches a certain threshold t. They also desire a switch with the ability to close a valve. But there is a difference in how both employees see the behaviour of the sensor. Employee A thinks that the sensor sends signal 00 when thetemperature is $\geq t$ and employee B thinks that the sensor sends signal 01 under the same circumstances. The two viewpoints are represented as follows, where square brackets $[\ldots]$ are used to indicate shared phenomena (in the sense of [ZJ97]) that occur at the interface between the software and its environment.

EMPLOYEE A:

$$\begin{array}{ll}
K: & temperature \geq t \Rightarrow [sensor\_00] \\
 & [door\_open] \Rightarrow boiler\_empty \\
 & [close\_signal] \Rightarrow valve\_closed \\
S: & [switch] \rightarrow [close\_signal] \\
 & [sensor\_00] \rightarrow [door\_open] \\
R: & temperature \geq t \Rightarrow boiler\_empty \\
 & [switch] \rightarrow valve\_closed
\end{array}$$

EMPLOYEE B:

$$\begin{array}{ll}
K: & temperature \geq t \rightarrow [sensor\_01] \\
 & [door\_open] \rightarrow boiler\_empty \\
 & [close\_signal] \rightarrow valve\_closed \\
S: & [switch] \rightarrow [close\_signal] \\
 & [sensor\_01] \Rightarrow [door\_open] \\
R: & temperature \geq t \rightarrow boiler\_empty \\
 & [switch] \Rightarrow valve\_closed
\end{array}$$

An overlap criterion could be based on the sharing, between formulae, of at least one literal. The discrepancy criterion for this example could be "identical sensor values should cause, and be caused by, identical conditions". Of course, this simple *ad hoc* criterion would not be useful, or should be refined, for other situations. But it also illustrates how a criterion could be defined without reference to the characteristics of the representation language, by focusing, for instance, on the properties of some artifact external to the software, like a sensor. With this criterion in mind, a discrepancy is found between S in A and S in B, as different sensor values lead to the same action (opening the door). The discrepant elements are:

$$\{\{[sensor\_00] \rightarrow [door\_open], [sensor\_01] \rightarrow [door\_open]\}\}$$

From this set, and following Phase I, a discrepancy in K is found:

$$\{\{temperature \geq t \Rightarrow [sensor\_01], temperature \geq t \rightarrow [sensor\_00]\}\}$$

but this discrepancy does not affect R, as both viewpoints agree about the requirements. So, in this example, the discrepancy belongs to the $K'S'R$ type, and its solution, according to table 1, should traverse these steps: Refutation (to solve the discrepancy in K consulting, for example, documentation about sensor behavior), transference (to achieve a non-discrepant K after the refutation has been done), checking (if it is needed to maintain internal consistency of K) and redesign (modifying S to adapt to the transferred K).

6 Conclusions and preliminary results

The process presented here has been validated in two case studies [Sil01]. The first of them dealt with the definition of a stock management system for a personal-computer builder and seller company. Two viewpoints were written using 40 sentences in natural language. Using informal overlap and discrepancy criteria, three discrepancies were found (one of the type $K'S'R'$ and two of the type $K'S'R$). The classification of the discrepancies pointed to the steps needed to solve them, according to table 1. The other example was expressed in Parnas tables similar to those used in the SCR method [PM95]. Two viewpoints offered discrepant views with regard to overriding a security injection system for a pressurized water reactor (this is a modification of the problem presented in [HJL96]). In total, 16 tables were used, and one discrepancy was found of the type $KS'R'$. From these cases it was showed that the process proposed is able to deal with different representation formalisms once adequate overlap and discrepancy criteria have been chosen.

In any case, the key idea of the method described in this paper is the need of a prior classification of the contents of each perspective *before* starting to look for discrepancies, in order to support the resolution. The classification we propose is based on the K, S and R categories introduced by Jackson [ZJ97]. This is an advantage over other viewpoint methods, given that different categories of knowledge are managed during requirements engineering, with independence of the particular formalism used to represent them. This previous classification leads naturally to a classification of the discrepancies found, and to adapt the resolution tasks to the special characteristics of each particular category affected by a discrepancy.

References

[HJL96] C.L. Heitmeyer, R.D. Jeffords, and B.G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, Jul. 1996.

[Jac95a] M. Jackson. *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices.* Addison-Wesley, Nueva York, 1995.

[Jac95b] M. Jackson. The world and the machine. In *17th. International Conference on Software Engineering ICSE'95*, pages 283–292. ACM, 1995.

[NER00] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging inconsistency in software development. *IEEE Computer*, 33(4):24–29, Apr. 2000.

[NKF94] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, Oct. 1994.

[Par95] D.L. Parnas. Using mathematical models in the inspection of critical software. In M. G. Hinchey and J. P. Bowen, editors, *Applications of Formal Methods*. Prentice-Hall, Englewood Cliffs, 1995.

[PM95] D.L. Parnas and J. Madey. Functional documents for computer systems. *Science of Computer Programming*, 25:41–61, 1995.

[SFT99] G. Spanoudakis, A. Finkelstein, and D. Till. Overlaps in requirements engineering. *Automated Software Engineering Journal*, 6:171–198, 1999.

[Sil01] A. Silva. *A Requirements Engineering Method for Discrepancy Handling*. PhD thesis, Facultad de Informática, Universidad Politécnica de Madrid, 2001.

[SM01] A. Silva and A.M. Moreno. A method for detection, classification and resolution of discrepancies in viewpoint-based requirements engineering. In *13th Intl. Software Engineering and Knowledge Engineering Conference (SEKE'01)*. Knowledge Systems Institute, 2001.

[Wit53] L. Wittgenstein. *Philosophical Investigations (Philosophische Untersuchungen)*. Blackwell, Oxford, 1953.

[ZJ97] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, Jan. 1997.