Reasoning about Inconsistency in RE
Separating the Wheat from the Chaff

Anna Zamansky[1], Irit Hadar[1] and Daniel M. Berry[2]
[1]*Information Systems Department, University of Haifa, Haifa, Israel*
[2]*Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada*

Keywords: Human Aspects of Software Development, Inconsistency Management, Requirements Engineering, Requirements Validation Formula.

Abstract: Inconsistency is a major challenge in requirements engineering. Traditionally, software requirements specifications (SRSs) are expected to be consistent, with the underlying assumption that this consistency is always achievable. However, with the growing complexity of software systems it has become clear that this assumption is not always realistic. This has led to new paradigms for inconsistency management, acknowledging that it is not only inevitable, but also even desirable at times, to tolerate inconsistency, even temporarily. However, for these paradigms to be widely accepted in industry, practicing software engineers must thoroughly understand the nature of inconsistency in SRSs and the strategies for its handling. This paper proposes a research agenda for preparing practicing software engineers to accept and successfully implement inconsistency management paradigms. As a first step in this direction, the paper describes an ongoing study in which we design an intervention into the perceptions of inconsistency for practicing software engineers. The intervention builds on teaching to them the Zave–Jackson requirements validation formula as an aid for analyzing the types of inconsistency they face, and conducting an empirical study of the effect of this intervention on their inconsistency management.

1 INTRODUCTION

Handling inconsistencies is a key challenge in requirements engineering (RE). Inconsistency in RE is typically described in the literature as a situation in which requirements or specifications contain conflicting or contradictory descriptions of the expected behavior of the system or of its domain. Such conflicting descriptions may come as a result of, for example, conflicting goals between different stakeholders, and changes introduced during the evolution of the requirements. (Nuseibeh et al., 2000; Nuseibeh et al., 2001). Traditionally, handling inconsistencies meant eliminating them all, with the underlying assumption that this elimination is always achievable. However, with the growing complexity of software systems it has become clear that this assumption is not always realistic. Indeed, over the last decades, a more tolerant approach toward inconsistency has evolved (e.g., Finkelstein et al. 1994; Nuseibeh et al. 2000; Finkelstein, 2000; Easterbrook and Chechik, 2001; Nuseibeh et al., 2001; Spanoudakis and Zisman, 2001; Ernst et al., 2012).

For a successful application of such approaches in industry, however, a thorough understanding of the notion of inconsistency by practitioners is required. Recent research (Hadar and Zamansky, 2015) demonstrates that this understanding is yet to be achieved. More specifically, practitioners identify inconsistency at times in cases of incompleteness or incorrectness, e.g., when an implementation does not meet the given set of requirements. The ability to distinguish inconsistency from other phenomena would enable effective inconsistency management in particular, and RE in general.

Research in RE has produced an explosion of ontological distinctions in requirements and their specifications. These distinctions matter for inconsistency management (Borgida et al., 2015), and therefore can be recruited for tackling the challenge of inconsistency management. One prominent example is the influential Zave–Jackson validation formula (ZJVF), which distinguishes Requirements *R*, Specifications *S*, and Domain Assumptions *D* (Zave and Jackson, 1997). Understanding the ZJVF provides:

1. a framework for partitioning a well formed software requirements specification (SRS) into its *R*, *S*, and *D* parts, and for validating that a system

Zamansky, A., Hadar, I. and Berry, D.
Reasoning about Inconsistency in RE - Separating the Wheat from the Chaff.
In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2016)*, pages 377-382
ISBN: 978-989-758-189-2

built according to *S* will result in *R* being met if the system is run in an environment in which *D* holds, and

2. a framework for reasoning about inconsistencies in an SRS that makes it very clear what the options are in order to resolve the inconsistencies, so that that the stakeholders can make rational decisions about the inconsistencies.

This paper explores the question of whether understanding the ZJVF and the ontological distinctions it imposes help an analyst to understand and manage inconsistency. Namely, it investigates how understanding the ZJVF and its ontological distinctions may affect analysts' decision making processes, enabling them to differentiate between inconsistencies and other failures, which are currently perceived as inconsistency but are actually some forms of incompleteness or incorrectness that need to be managed differently from inconsistencies. In addition, the paper proposes a research agenda for finding means to guide analysts' understanding of inconsistency in RE and for empirically evaluating these means. We start with examining the ZJVF.

The next section provides details on the ZJVF and discusses it in the context of inconsistency management in RE. Section 3 presents an outline of experiments we plan to conduct for evaluating the contribution of the ZJVF for distinguishing between inconsistency types and between inconsistency and incorrectness or incompleteness, and deciding on the appropriate strategies to manage them, based on this distinction. Section 4 discusses the potential contribution of the proposed research agenda and future research steps.

2 THE ZAVE-JACKSON VALIDATION FORMULA AND INCONSISTENCY MANAGEMENT

The ZJVF assumes that part of the world has been divided into an environment, *Env*, and a system, *Sys*, that overlap, i.e., intersect, at their interface, *Intf*, as is shown in Figure 1. The *Env* is the part of the world that is affecting and is affected by the *Sys*, and the *Sys* is the computer-based system (CBS) that is desired by the customer who provides the requirements. The elements of the ZJVF, $D,S \vdash R$, are the assertions *D*, *S*, and *R*:

- *D*, domain assumptions, is what about the *Env* that the *Sys* is allowed to assume in its execution in the *Env*; it is written in the vocabulary of the *Env*.
- *S,* specification, is a description of the behavior of the *Sys*; it is written in the vocabulary of the *Intf*, which is the vocabulary shared by the *Env* and the *Sys*.
- *R*, requirements, is a description of the customer's requirements for the *Sys*; it is written in the vocabulary of the *Env*.

The formula says that the *Sys* is validated to meet its requirements in the *Env* if given the holding of *D*, *S* is sufficient to entail *R*.

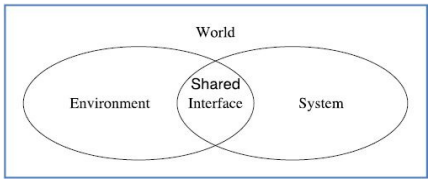

Figure 1: The ZJVF Worldview.

A key point of the ZJVF framework is that a well formed SRS has all of *D*, *S*, and *R*, and each sentence in the SRS is identifiable as being one and only one of *D*, *S*, and *R*. Typically, *D* is given in indicative mood in present tense in sentences about the *Env*; *S* is given in optative mood in sentences about the *Sys* using the verb "shall"; and *R* is given in optative mood in sentences about the *Env* using the verb "will". There are other ways to make the distinctions clear.

It should be noted that in the formula $D,S \vdash R$, each of *D*, *S*, and *R* is an assertion written in a formal language so that the required proof can be carried out. *S*, being about a program, written in a human-made language, is formal in that it is possible to carry out a formal proof that a program *P* satisfies *S*. However, as statements about the real world, each of *D* and *R* is inherently informal, because we are never certain about our knowledge of the real world, and what we believe to be true about the real world is constantly changing.

Moreover, each of *D*, *S*, and *R*, being an assertion, can be either *consistent* (*satisfiable*) or *inconsistent* in the classical logical sense. For example, the logical formula $A \vee \neg A$ is consistent, but $A \wedge \neg A$ is inconsistent.

Each of *D* and *R*, being an assertion about the real world, if consistent, can be *correct*, *incorrect*, or *undetermined*. An assertion *A* is correct if *A* is known to accurately describe the real world. *A* is incorrect if *A* is known to not accurately describe the real world[1]. *A* is undetermined if whether *A* is correct or incorrect is not known. For example "The earth is a sphere" is

[1]Note that "incorrectness" can be considered as inconsistency with the *Env*. Hence, there is a logic to calling all sorts of problems with an SRS "inconsistency".

correct, "The earth is a disk" is false, and "There are gravity waves" was undetermined prior to 2016. However, as of 11 February 2016, the world learned that scientists had determined that it is correct (Cho, 2016).

Thus, as opposed to consistency, correctness is not a logical question, but an empirical one, which can be answered only by observation. The job of science is to determine whether a logically consistent assertion, a.k.a. a theory, is correct. Experiments plus statistical analysis of the data generated during the experiments allow us to determine to whatever degree of certainty we need, whether a theory is supported by the data and thus whether the theory as an assertion about the real world is correct. Even something regarded as correct at some level of detail may not be correct at another level of detail. For example, "The earth is a sphere" is correct for most purposes, but for other purposes, it is incorrect, and instead, "The earth is an oblate spheroid" is correct.

Finally, *S* can be *OK* or *not OK* with respect to *D* and *R*. *S* is OK w.r.t. *D* and *R* if $D,S \vdash R$ and is not OK, otherwise. In other words, an *S*, specifying a program is OK if it causes *R* to be met when *D* holds, even if in another *D'*, *S* may not be OK.

Using the above terms, the ZJVF is based on two assumptions, which are crucial for our purpose:

1. Each of *D*, *S*, and *R* is consistent, so that it is possible for *D* and *S* to be correct and *S* to be OK.
2. *D is* correct in the *Env*, and *R can be* correct in the *Env*; that is, *D* accurately describes the existing *Env*, and there is nothing in the world that would prevent *R* from accurately describing a future *Env*.

Each way that one or both of the above assumptions does not hold gives rise to different types of what is called "inconsistency" in the literature cited in Section 1.

The following simple examples demonstrate the different types.

Consider the following requirements:

*R*1. The GPA of every student will be reported.

*R*2:

*r*2a. The GPA of every student will be reported.

*r*2b. The GPA of any student who has not taken any courses will not be reported.

*R*3:

*r*3a. The GPA of every student who has taken at least one course will be reported.

*r*3b. The GPA of any student who has not taken any courses will not be reported.

*R*4. A useful, accurate letter of recommendation will be generated *automatically* (i.e., with artificial intelligence (AI)) for each student.

*R*1 is consistent. *R*2 is inconsistent. *R*3 is consistent, and R4 is consistent.

*R*1 can be correct. *R*2 cannot be correct, because it is inconsistent. *R*3 can be correct. *R*4 cannot be correct, because automatically generating useful, accurate letters of recommendation is beyond the capabilities of software.

Let us now consider the following domain assumptions:

*D*1. Each student takes at least one course.

*D*2. Each student takes zero or more courses.

*D*1 is consistent. *D*2 is consistent

*D*1 can be correct. *D*2 can be correct. If *D*1 is correct, then *D*2 is correct, but not necessarily the reverse. If *D*1 is incorrect, then *D*2 can still be correct.

Finally, consider the following specifications:

*S*1. The program shall report a student's GPA as the sum of the grades in her courses divided by the number of courses she took.

*S*2. For a student who took at least one course, the program shall report the student's GPA as the sum of the grades in her courses divided by the number of courses she took. For a student who took no courses, the program shall report that the student's GPA is not defined.

*S*1 can be OK if it is never asked to compute the GPA of any student who took no courses. *S*2 is OK regardless of the number of courses taken by the student, whose GPA is to be computed.

Let us systematically explore various SRSs with these Ds, Ss and Rs, in each case starting with an *R*, observing a *D*, and then picking an *S*. With each, we describe the conclusion that an analyst should draw and what his or her resulting course of action should be.

Taking *R* as *R*1

Suppose that *D*1 is correct. Then, *D*1, $S1 \vdash R1$, and *S*1 is OK, even though *S*1 can be not OK, because given *D*1, *S*1's not OK state cannot occur.

Suppose, on the other hand, that *D*2 is correct. Then, it is not the case that *D*2, $S1 \vdash R1$, in particular because *S*1's not OK state can occur. Also, if indeed there are students who have not taken any courses, their GPA cannot be reported as required by *R*1. In this case, the customer must be asked to choose between two responses:

1. If students who have not taken any courses are extremely rare, then *tolerate*[2] that *D*2 is incorrect and treat the student who has not taken any courses manually. This toleration is achieved by replacing *D*2 by an admittedly incorrect *D*1 and

falling back to $D1$, $S1 \vdash R1$.

2. Weaken $R1$ to an R that does not require that a GPA be reported for all students and then have software that checks for a student's taking zero courses, such as that specified by $S2$.

In this example, the customer knows that a student not taking any courses is not rare at all and that it is better that the software handles the case. Therefore, the customer chooses to weaken R and to use software specified by $S2$.

Taking *R* as *R*2
An initial attempt to weaken R leads to $R2$, which is inconsistent. Fixing the inconsistency leads to $R3$.

Taking *R* as *R*3
Suppose again that $D2$ is correct. Then, $D2$, $S2 \vdash R3$. Interestingly, $S2$ is sufficiently robust that by itself, $S2 \vdash R3$. That is, the distinction between $D1$ and $D2$ is irrelevant.

Taking *R* as *R*4
No S can entail $R4$, because, as mentioned above, it is simply not possible to algorithmically generate useful, accurate letters of recommendation. Thus, the customer must be told that his or her requirement cannot possibly be met and to find another requirement. A possible achievable requirement is to print human-composed letters of recommendation that are already stored in a database.

The above examples demonstrate how the analysis of the RE problem through the lens of the ZJVF induces different strategies for inconsistency management. While it may be sensible, and even desirable in some cases, to tolerate incorrectness, inconsistency in an SRS is an indication of an inherent, serious problem. At the very least, the analyst discovering the inconsistency must make a memo to the project that the inconsistency exists so that all are aware of it and can take it into account in their work. In this way, the inconsistency is tolerated temporarily during an investigation into ways to resolve the inconsistency. Ultimately though, the inconsistency must be resolved before the SRS can be considered as delivered. That resolution may involve tolerating incorrectness, changing the software, or abandoning or changing a requirement. All these activities can be regarded as *managing an SRS's inconsistency*, in the more general sense of the word "inconsistency".

[2]Admittedly, for such a small, easily fixed problem, this choice is clearly preposterous. However, in real life, there are statistically rare situations that no man-made system can handle correctly in every case for which a decision to tolerate the failures that will result in the situations is a reasonable choice. An example is the possibility that a flying aircraft will hit a bird and crash.

3 THE EXPERIMENT

Previous research has shown that practitioners include in their personal definitions of inconsistency cases of incorrectness (Hadar and Zamansky, 2015). For effective inconsistency management to be viable, it is important that requirement analysts would fully understand the difference between inconsistency and incorrectness. We believe that a requirement analyst who understands the ZJVF will make better distinctions between inconsistency and incorrectness, leading to better strategies for handling these cases. Therefore, we propose an intervention, that is, teaching requirements analysts the ZJVF and how to use it for identifying and understanding inconsistencies in SRSs. Balaban et al., (2014) have shown how teaching patterns to UML modelers led to the modelers' producing better quality UML models. We plan to employ a similar strategy here.

Thus, we propose empirical research addressing the research question (RQ):

> Does teaching a requirement analyst the ZJVF improve his or her ability to identify and deal with inconsistencies in SRSs?

To answer this RQ, we are considering an experiment outline as described below.

The planned experiment is a pre-test–post-test experiment, with the intervention occurring between the tests (Shadish et al., 2002).

Pre-test:

a. The test presents examples of SRSs, in which inconsistencies and incorrectness may exist. While each SRS is divided into its *D, S*, and *R* parts, the sentences of the SRS are not directly labeled as to which of *D, S*, and *R* it is part of. Rather, linguistic clues of some kind are provided e.g., "is" vs. "shall" vs. "will".
b. For each of these SRSs, the test asks if there is a problem in the SRS, and if there is, what it is and what should be done to solve it.

For the pre-test, it is hypothesized that most participants will not distinguish between inconsistency and incorrectness, and between different types of inconsistency, leading to proposed strategies that may be ineffective or irrelevant to the problem at hand, including those that indicate that they have no clue as to what is going on.

Intervention: Teaching the ZJVF, with emphases on the distinction between *R*, *S*, and *D* and on how to use the formula to find problems in an SRS and to suggest resolutions to these problems.

Post-test: The test is the same as the pre-test.

For the post-test, however, it is hypothesized that the participants will demonstrate an enhanced ability to distinguish between different cases calling for different solutions, that provide answers that are well focused as the example analyses given at the end of Section 2.

At the time of this writing, we are running a series of pilot studies of this experiment in order to

- refine the SRSs used in the experiment,
- refine the questions asked of each SRS,
- determine suitable hypotheses to test, and
- determine suitable measures of the participants' responses that will allow testing the hypotheses.

4 INITIAL LESSONS LEARNED

From a pilot study already conducted, we have learned several important lessons about conducting the planned experiment and considerations to be taken in order to achieve valid and insightful findings.

It is hard to create good SRSs for this purpose; each SRS should be

- small enough to be dealt with in the few minutes that are realistic for the short duration of a controlled experiment,
- complex enough to have real problems that can be found, but
- not so complex that there is a good chance that participants will not find the problems.

It is hard to judge the computer-science background of the participants, the background that is needed to know

- the kind of defects that software can have,
- the limits of what software can do,
- the limits of what current hardware can do.

This information is needed to know whether it is reasonable to expect participants to be able to find some of the inconsistencies and incorrectnesses that can be included in a proposed SRS.

It is hard to compose questions about an SRS; each question should

- not reveal so much that someone can guess the answer without at least an intuitive understanding of the ZJVF, but
- not be so open ended that there is a good chance for irrelevant answers.

It is hard to teach the ZJVF in the short period that is realistic for the short duration of a controlled experiment, to teach it well enough that participants are able to apply it in order to correctly analyze SRSs and resolve their inconsistencies and incorrectnesses.

Finally, it is hard to motivate student participants to answer post-test questions with other than "the same as in the pre-test".

5 DISCUSSION AND FUTURE WORK

It is clear that inconsistency management should be explicitly addressed in RE and techniques for doing so should be included in the toolbox available to requirements analysts.

This paper suggests that understanding the ZJVF allows a requirements analyst to identify the sources of a problem, i.e., an inconsistency or incorrectness, in an SRS, and to know what needs to be done to resolve the problem. If the problem is an incorrect SRS, the analyst can fix it directly. If resolving the problem requires a decision from the customer, the analyst knows what options can be offered to the customer for the resolution.

The proposed research agenda can be seen as complementary to the various theoretical frameworks proposed for inconsistency management in RE, toward bridging the gap between theory and practice. More generally, the research is situated in the milieu of research on how effective are the techniques in the SE toolbox, e.g., as did Balaban et al. when they empirically tested whether their pattern-based approach for identifying abstractions helped students make better UML class diagrams.

REFERENCES

Balaban, M., Maraee, A., Sturm, A. and Jelnov, P., 2014. A pattern-based approach for improving model quality. *Software & Systems Modeling*, Online: DOI 10.1007/s10270-013-0390.

Borgida, A., Jureta, I. and Zamansky, A., 2015. Towards a general formal framework of Coherence Management in RE. In Proceedings of the 23rd *IEEE International Requirements Engineering Conference* (pp. 274–277) IEEE.

Cho, A., 11 Feb. 2016. Gravitational waves, Einstein's ripples in spacetime, spotted for first time. *Science Magazine*, http://www.sciencemag.org/news/2016/02/gravitational-waves-einstein-s-ripples-spacetime-spotted-first-time.

Easterbrook, S., and Chechik, M., 2001. A framework for multi-valued reasoning over inconsistent viewpoints. In

Proceedings of the 23rd *International Conference on Software Engineering (pp. 411–420) IEEE*.

Ernst, N. A., Borgida, A., Mylopoulos, J. and Jureta, I. J., 2012. Agile requirements evolution via paraconsistent reasoning. In Advanced Information Systems Engineering (pp. 382–397) *Springer*.

Finkelstein, A., 2000. A foolish consistency: Technical challenges in consistency management. In *Database and Expert Systems Applications* (pp. 1–5) Springer.

Finkelstein, A. C., Gabbay, D., Hunter, A., Kramer, J., and Nuseibeh, B., 1994. Inconsistency handling in multiperspective specifications. *IEEE Transactions onSoftware Engineering*, 20(8), 569–578.

Hadar, I. and Zamansky, A., 2015. Cognitive factors in inconsistency management. In Proceedings of the 23rd *IEEE International Requirements Engineering Conference (pp. 226–229) IEEE*.

Nuseibeh, B., Easterbrook, S. and Russo, A., 2000. Leveraging inconsistency in software develoment. Computer, 33(4), 24–29.

Nuseibeh, B., Easterbrook, S and Russo, A., 2001. Making inconsistency respectable in software development. *Journal of Systems and Software*, 58(2), 171–180.

Shadish, W. R., Cook, T. D., Campbell, D. T., 2002. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Boston: Houghton Mifflin. ISBN 0-395-61556-9.

Spanoudakis, G. and Zisman, A. (2001). Inconsistency management in software engineering: Survey and open research issues. *Handbook of software engineering and knowledge engineering*, 1, 329–380.

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