# Do You Speak Z? Formal Methods under the Perspective of a Cross-Cultural Adaptation Problem

Andreas Bollin
Software Engineering Research Group
Alpen-Adria Universität Klagenfurt
Klagenfurt, Austria
Email: Andreas.Bollin@aau.at

***Abstract*—The use of formal specifications seems to be a silver bullet in a world where technical systems become more and more software intensive and where quality considerations become increasingly important. However, formal methods and the use of formal specifications are by far not so widespread as they should and could be.**

**This position paper argues that a broader view onto this situation can be very helpful. It introduces the formal development process as a cross-cultural adaptation problem, discusses pros and cons, and, finally, comes up with a refined model for a formal software development process.**

## I. Introduction

With today's companies and development teams no longer confined to single locations, much work is undertaken in global teams. But, even when the development is limited to a single location, size (and complexity) of projects necessitates a lot of different departments and stakeholders to be involved. When people from different environments and (cooperate) cultures communicate, they bring with them their cultural knowledge and background [1], which implies that they speak and interpret the communication of others from their cultural perspective [2].

Formal methods are intended to provide the means for greater precision in both, thinking and documenting the preliminary stage of the software creation process. However, with the steadily growing use and with the growing sizes of formal specifications, their development can not be done by a single person anymore. So, different teams members and persons are involved – ideally including the customers.

Taking a closer look at this situation, it is very likely that not all stakeholders are able to speak and think in the same technical terms, and it is also very likely that the stakeholders have different cultural backgrounds and have different communication styles [3]. Anawati and Craig [4, p.45] therefore explain how easily this can lead to team members struggling with cross-cultural communication, as they have not considered their differences nor considered that this may affect the projects' performances. As one solution they suggest a framework for adaptations, but only focus on behavioral aspects.

The problem of cross-cultural adaptation is not a new one, and since decades especially study design authors come up with a series of models to overcome the problem of adapting measures for the use in other than the source language [5], [6], [7]. Basically, all the suggested models make use of cycles of translations and re-translations, yielding, within just a small number of iterations, translations that are operationally and semantically similar.

This position paper now tries to argue for the explicit use of a cross-cultural adaptation model in the development of formal specifications. The reasons are manifold and will be explained in more details below. However, one argument for such a use is that in the diversity of the different stakeholders there is a benefit. By trying to match their different conceptual models of the system to be built, differences are detected. And, dealing with these differences (a) can help in improving the overall quality of the specification and (b) can also help in raising the acceptance of the use of formal methods.

The paper is structured as follows: Section 2 briefly introduces the main ideas behind cross-cultural adaptation models. Section 3 deals with stumbling blocks and impediments when working on (and with) formal specifications. Section 4 comes up with a formal specification development model following a cross-cultural adaptation model, and Section 5 concludes the paper with an outlook of future work that has to be done.

## II. Cross-Cultural Adaptation

According to Young Yun Kim's work [8] in 1988 (which is based on previous work of Milton Gordon [9] and Robert Ezra Park [10]), the concept of cross-cultural adaptation refers to a process in and through which a person achieves an increasing level of psychological and functional fitness with respect to the receiving environment. Adaptation is seen as a natural process as all human beings are born into an unfamiliar environment and are therefore forced to become part of a culture (enculturation). In his intercultural communication course, John Baldwin [11] summarizes the main idea behind Kim's work as a systematic approach, where an individual can be seen as a "system" and where changes in the environment lead to a lack of balance. The “system” then tries to get back to balance, either by changing some part of the environment or by adjusting to it.

In the field of technical and medical sciences, the term “cross-cultural adaptation” is used, according to Beaton et.al. [12, p.3186], to encompass a process that looks at both, language (with respect to translation) and cultural adaptation issues in the process of preparing a questionnaire for use in another setting. There are several scenarios where adaptations should be considered, and Guillemin [13] suggested five different situations to be prepared for. The first scenario is a situation, where the same languages and the same cultures are involved. Then no adaptation is necessary. On the other hand, the last scenario encompasses situations, where culture, language and country are different, necessitating translation and cultural adaptation. Beaton et.al. suggest a model consisting of six stages [12, p.3188], and as this model is some kind of “basis” for the model introduced in Sec. IV it is presented here briefly:

1) Stage I: Translation. At first, a forward translation by two translators is conducted. The first translator should be aware of the concepts in the questionnaire, the second translator should be a a native one, neither aware of the concepts in the text nor having any background in this area.
2) Stage II: Synthesis of the Translation. Here, the translators and an observer sit down together and produce a synthesized version of the questionnaire. All the discrepancies are recorded.
3) Stage III: Back Translation. Totally blind to the original version, a translator then translates the new version back. A report is created. This step can be seen as a validity check (but, a match between the original version of the questionnaire and the back-translated version does not guarantee a satisfactory forward translation). Two of these back-translations are considered as minimum.
4) Stage IV: Expert Committee. The role of the committee is to consolidate the different versions of the questionnaire and to develop the pre-final version. For this, they are reviewing all the reports and try to achieve the necessary equivalence between the source and target version of the questionnaire.
5) Stage V: Pretesting. Subjects are asked to complete the questionnaire. They are interviewed to probe out what he or she thought was meant by each item. The responses are compared to the intended meaning.
6) Stage VI: Submission and Appraisal of all reports. At this final stage all the forms and reports are sent back to the developers.

For the purpose of this work, we can look at the *cross-cultural adaptation process as an effort in achieving a semantic, idiomatic and conceptual equivalence between a source and a target document*. It is important to note that the process tries to produce equivalence between source and target based on content.

On the first sight, the development of a questionnaire might not have so much in common with the development of a formal specification. However, when developing a system in an inhomogeneous environment with different stakeholders involved, the similarities become clear. Customers usually do not speak the same (technical) language as the developers and might also come from different cultures; the development teams might consist of multi-national team members and might also be distributed across the world; development teams might have different cultures (idioms, styles, notations), too, and the same holds for the persons at the different management hierarchies. What can be seen as a mere management problem of the project lead is in fact also a cross-cultural problem to be bridged. And, dealing with this issue, also helps to overcome some challenges with formal specifications.

## III. Formal Development Challenges

As already discussed by Bollin [14], formal specifications are no documents that are written once and that should be written just at the beginning of the software development process. A specification is not per se and quite from the beginning a “correct” mapping of the requirements. It needs time to create a first, useful version and then a lot of effort (and revisions) to develop a specification that is close to the concepts the customers (and/or developers) have in mind. This implies that our formal specifications are constantly changing during the software development phases, they evolve, and the effect of evolutionary changes can also be measured [14], [15].

This section now takes a closer look at problems and challenges that a developer has to deal with when working on and with a formal specification. The list of challenges stems from observations during 17 years of working with and 13 years of teaching formal specification languages.

### A. Logic

Already in 1987, Hoare [16] stated that the development of any complex system is likely to require the use of multiple notations at different stages in the process and to describe the different aspects of a system at various levels of abstraction. Specifications are used to write down statements of properties required by a product, or a set of products, but there is certainly a trade-off between the specification language and the levels of abstraction that it supports [17]. Very common is the use of the propositional and predicate calculus, and some specification languages may have “wider vocabularies” and constructs to support expressing our statements (and allowing for more freedom in expressing properties), while other languages are more limited in this respect. However, be an expression more or less abstract, the developer is at the end forced towards a particular implementation. This implies that the developer is hopefully interpreting the statement correctly.

Let the following specification in the specification language Z [18] be given:

$$\begin{array}{|l} \mathit{DivByThree} : \mathbb{P}\,\mathbb{N}_1 \\ \hline \forall x : \mathbb{N}_1 \mid x \bmod 3 = 0 \bullet x \in \mathit{DivByThree} \end{array}$$

This specification provides the signature of an identifier called *DivByThree*, telling us that is represents a set of natural numbers. The predicate below the line restricts the elements of this set. Now, when asking developers (and students) of how a valid set *DivByThree* is looking like, answers usually consist of sets that look as follows: $\{3, 6, 9, 12, ..\}$. When then asking whether the set $\{3, 6, 7, 8, 9, 10, ..\}$ is a valid representation of *DivByThree*, then the developers (and students) are up in arms against this set.

Well, the set $\{3, 6, 7, 8, 9, 10, ..\}$ is a valid representation, though. The reason is that the restricted quantifier (citing a developer: "which is so easy to read: for all elements x which are dividable by three, x is in the set") is to be resolved as follows:

$$\begin{array}{|l} \mathit{DivByThree} : \mathbb{P}\,\mathbb{N}_1 \\ \hline \forall x : \mathbb{N}_1 \bullet x \bmod 3 = 0 \Rightarrow x \in \mathit{DivByThree} \end{array}$$

That means that in the case when the number is divisible by three it is in the set, but the specification says nothing about the case when the number is not divisible by three. Such a number might be in the set or not. So, a first challenge that might lead to unwanted side-effects (in the implementation or communication) is the following:

C1 Mistakable logical expressions. Nothing is more logical than a logic expression, but mathematical logic is not always "equivalent" to human logic.

The example above is quite simple, but one can imagine that more complex logical expressions easily end up in total misunderstandings. One of the rare, empirical studies in this field, conducted by Vinter, Loomes and Kornbrot [19], also showed that there are mathematical constructs (like implication and negation) that result in misunderstandings and, as a result, even in erroneous specifications itself.

## B. Requirements

The advantages of formal specifications not only lie in verification and validation considerations. They form the basis for incrementally pinning down the most important requirements (that are later on subject to change during ongoing refinement steps).

Collecting and correctly writing down requirements is a field of research on its own and by far out of the scope of this paper. Van Lamsweerde brings it to the point when he states that "requirements engineers live in a world where inconsistencies are the rule, not the exception" [20, p.88].

However, there is always the chance that there are cultural and language-specific differences that can not be detected and resolved easily by reviews, prototyping or test case generation (which are the standard techniques for such cases). At least the following challenge should be mentioned:

C2 Different interpretations of a requirement. A requirement can be written down clearly, but, due to national or cultural differences, there is the chance that it is interpreted in a totally different manner.

For example, let us assume that a customer tells us to let the system "close the entrance door exactly at *half six*". This might be a clear statement that can be formulated as a predicate. A friend of mine from NASA and I sat together at the ICFEM'11 conference, and we were both confused about such a statement. "Half six" means 5:30pm for an Austrian developer, and 6:30pm for an American one. Once written down as 5:30pm by a developer, such a misunderstanding is hard to detect without appropriate feedback loops.

Another example influenced by culture and nation would be the the use of colors which have different meanings in different cultures.

Cultural differences are not the only limiting factor, also languages have their limitations, leading to another challenge:

C3 Unformalizible or counter-intuitive statements.

Different languages are differently suitable for expressing facts. At the end, we have the problem of writing down expressions in natural language in a formal manner. And here, especially sentences with plurals make problems when writing them down in first order logic.

Most well known is the Geach-Kaplan sentence "some critics admire only one another". Boolos [21] (attributing it to Kaplan) proves that it is nonfirstorderizable by showing its second-order translation to be true in every nonstandard model of arithmetic but false in every standard one. Developers prefer singular quantifiers over plural quantifiers, and thus again some of the requirements might be pinned down (or be communicated) in a misleading manner.

## C. Comprehensibility

Both sections above, *Logic* and *Requirements*, of course go hand in hand with the issue of comprehensibility. Nevertheless, they look at the problems at a fine-granular level, namely at the level of a single statement or property.

Only comprehensible statements are useful in the software development process (be it as basis for refinement or test case generation, be it as an additional source of documentation), and so, on a larger scale, one ends up with the same questions: how comprehensible is a given specification?

Writing a "good specification" comes with practice, though there exist some guidelines. For example, the recommendations of Gravell [22] are inter alia:

- prefer clarity to brevity,
- give an implication its natural order, or avoid using implications entirely,
- narrow the syntactic gap by writing mathematical definitions that follow the structure of a natural, readable description,
- where the mathematical idiom is commonly understood, use it,
- composition of functions (or relations) can be shorter and simple than set comprehension

The main message behind Gravell's list of recommendation is to keep the specification as simple as possible. He also

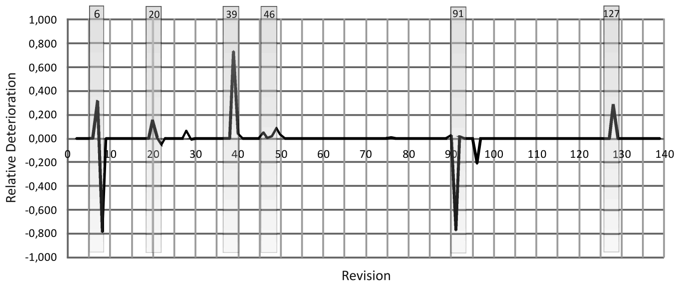

Fig. 1. Relative change of the size of deterioration for 139 revisions of the *WSDL* specification.

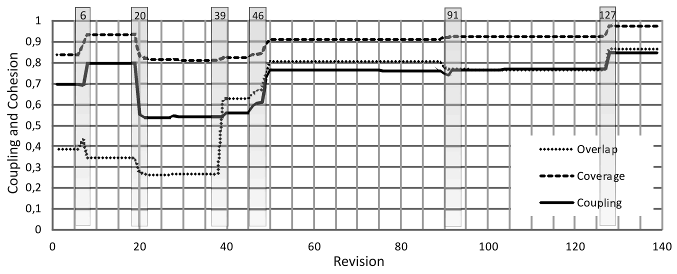

Fig. 2. Values of *Coupling*, *Overlap* and *Coverage* for 139 revisions of the *WSDL* specification.

states that the key point is that a specification should include mathematical and English descriptions of the system which are in clear agreement. However, in a "postscript" he provides, among other variants, the following two specifications of the set of prime numbers:

$$\begin{aligned} &\mathit{primes}_1 == \{n : \mathbb{N} \mid n \leq 2 \land \\ &\qquad\qquad \neg\,(\exists\, m : 2 \mathrel{..} n \bullet n \bmod m = 0)\} \\ &\mathbb{N}_2 == \mathbb{N} \setminus \{0, 1\} \\ &\mathit{primes}_2 == \mathbb{N}_2 \setminus \{n, m : \mathbb{N}_2 \bullet n * m\} \end{aligned}$$

The first definition ($\mathit{primes}_1$) is quite close to the natural language definition of prime numbers, whereas the second definition ($\mathit{primes}_2$) states that a prime is not a product of a natural number larger than 1. Gravell admits that it is up to the reader to decide whether a shorter and clearer specification is also a better specification.

Existing style guides and templates deal with the issue of comprehensibility, but they also lack in the broader perspective. Recent advances in the measurement of formal specifications [23] [24] point out that the overall complexity and quality of a specification really matters. So, another challenge should be added to make up the full picture:

C4 Too complex specifications and specifications of low quality.

A lot of concepts that we have to deal with are inherently complex. However, there are very often different ways in expressing these concepts. And specifications are of less complexity and thus of higher quality when separate thoughts are not unnecessarily interwoven. Let us look at an example.

The Web-Service Definition Language (*WSDL*) 2.0 [25], which is an XML language for describing Web services, has been specified by making use of Z, and Figure 1 presents a measure called *relative deterioration*, applied to the 139 different revisions of the specification [14]. Relative deterioration compares two successive versions of a specification and it is greater than zero when the number of separate concepts within a schema decreases. It is negative, when the number of separate concepts handled by a single schema raises, indicating some probably unintentional deterioration of it. (The measure might be debated on, but it is nothing else than an indicator for the isolation of concerns in schemas.)

During the development a lot of change requests occurred, leading to new revisions. The astonishing thing is that the development strategy (in the documentation called "last calls") behind the specification is visible in the figure. Usually, a change is triggered, first, by a structural improvement (to be seen as a positive amplitude of the measure in the diagram), followed by the introduction of the new thought which in some of the cases resulted in a negative amplitude of the measure. This approach can be seen in Fig. 1 at revisions 6 and 7, 20 and 21, or 90 and 91.

Changes in the specification also have an influence on quality measures. For a larger study, the measures of coupling and cohesion have been mapped to formal Z specifications [23], [15], and Fig. 2 presents the values of *Coupling*, *Overlap* and *Coverage* for the 139 revisions of the *WSDL* specification.

The values for *Overlap* are, for example, low at the beginning (around 0.39 at revision 1). This indicates (and can also be seen in the specification text of the *WSDL* specification) that a lot of concepts or "thoughts" within the specification schemas are unrelated. Even more, a lot of new concepts have been introduced later on, and only when the developers (due to a series of change requests) started to refactor the specification (from revision 39 on), the trend stopped and the values started to increase again. Similarly, the values of *Coupling* or *Coveral* are strongly influenced by the different changes in the specification.

## D. Notation

Communication is an essential aspect in every project. However, it is rather the rule than the exception that not all stakeholders do speak the "same language" – especially in a technical sense. Compared to specifiers, developers might be more firm in a programming language, and managers might have problems with both, specifications and software code. Moreover, customers might even interpret the simplest drawings incorrectly. So, the last challenge is therefore related to the notations used:

C5 Misleading and hard to understand notations.

Even when fading out cultural and national differences, people in a project do speak different languages and, according to their profession, do make use of different notations.

Though one could debate on the semantics of notations that have been introduced during the past decades, the common use of a standard set of notations (and languages) is an improvement in every project. On contrary, even standardized notations have their limits, they might either be so rich in their syntactical elements that again only a few people are able to communicate with it, or they are so restricted to a single domain that non-standardized (and thus again hard to communicate) extensions have to be used.

### *E. Translation as a Necessity*

When going again over the challenges C1 to C5 briefly, we can notice that every item can be dealt with by translating it to another form (either another language or another representation):

- Logical expressions and requirements (C1, C2 and C3) can be reformulated and paraphrased in natural language. For gaining a deeper understanding of their meaning they can also be re-written.
- Complexity and quality can be monitored (C4), resulting in specifications with measures assigned to them. Basically this means to map a specification to a set of numbers that experts are able to interpret.
- Hard to understand (or very complex) parts (C5) can be transformed to simpler representations, eventually abstracting away from irrelevant aspects.

When not taking care, our specifications are hard to understand and do have errors in them that cannot be detected automatically. It is the belief of the author of this paper, that such problems can, if ever, only be detected by making use of measures, by translating them to different notations and by an ongoing discussion/reflection with the different stakeholders. In any case, we have to *translate* the specification into *another* language (notation), not forgetting the constraint that the readers of the translated versions have different cultural/national and educational backgrounds. The integration of a cross-cultural adaptation model comes naturally to mind.

## IV. Formal Specification Cross-Cultural Adaptation Model

The basic idea behind the cross-cultural adaptation models of Beaton et.al. [12] is to find a balance (in respect to semantic, idiomatic and conceptual properties) between (two) different documents. This balance is found by a translation loop, making use of forward and backward translation steps. The source document is translated into two versions of the target document, these two versions are then synthesized and translated back to a "source" document. By comparing the differences between all the versions and by looking at the translation process itself, the chance is high to reach high semantic, idiomatic and conceptual equivalence.

### *A. Adaptation Model*

When applying cross-cultural adaptation principles to the formal development of software, one has to decide which

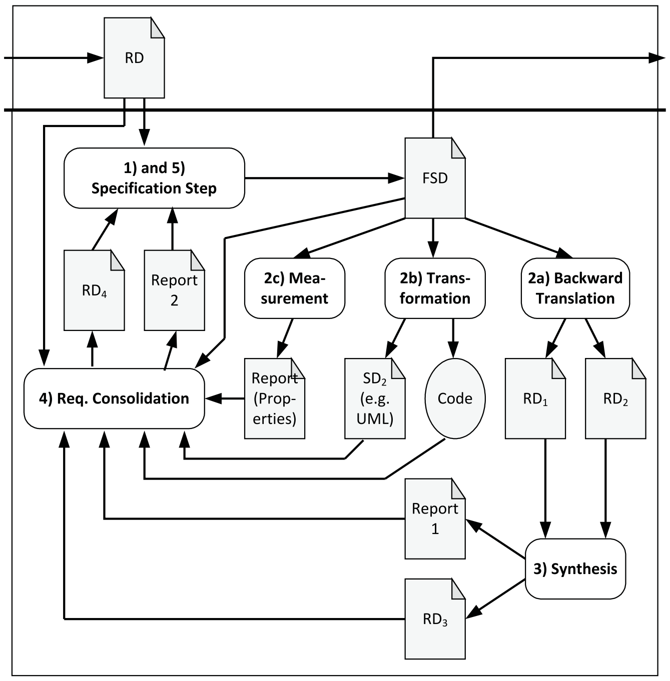

Fig. 3. Part of a formal methods development process with a focus on cross-cultural adaptation steps (RD .. Requirements Document(s), FSD ... Formal Specification Document(s), SD ... Specification Document(s)).

types of documents to take as source, and which as target. For reasons of simplicity and for keeping the effects on existing development processes at a minimum, it is assumed that a requirements document and a first version of a formal specification are already available. With this, the source document is the first version of the formal specification, and the target document is a new version of the requirements document (which is, in turn, again the source for a new version of the formal specification). Fig. 3 provides an overview of the most important steps:

1) and 5) *Specification Step*. In this first (and fifth) step it is assumed that domain and specification experts, based on all available documents, are working together in order to produce a first (or improved) version of the formal specification document.
2) *Translation Step*. Basically, this step is split into three sub-steps:
   a) *Backward Translation*. The formal specification is translated back to at least two requirement documents. This is done by two different translators, one with domain knowledge, and one without domain knowledge. As formal specifications are quite often embedded in natural language text describing the specification parts, it makes sense (in such a case) to cut out these text passages for at least one of the translators.
   b) *Transformation*. The formal specification is mapped to another type of representation. This could be done by a transformation to UML diagrams (as can e.g. be

done in an optimized way for Z [26]) or animations (as done by the Alloy framework [27]), or even by the generation of code (like in VDM [28]). An expert in this field is only necessary when this step can not be done automatically or when the automatic transformation does not result in "clearly readable" documents.

c) *Measurement*. Here, the formal specification is "translated" into a set of attributes, describing relevant properties of it. Again, usually this step can be done by tools[1].

3) *Synthesis*. The output of step 2a) consists of 2 new versions of the requirements document. Totally blind to the original requirements document, the two new versions are merged together, and a protocol is produced.
4) *Requirements Consolidation*. This step is the most complex one. The different stakeholders (eventually also the customer) sit together and take a look at the reports (measures, synthesis), and analyze the differences between the source documents (old requirements document, old formal specification) and the new requirements document. They eventually make use of the alternative specification documents from step 2c) and the animations produced by step 2b). The result is a report (including a change log) and a new version of the requirements document. These documents are then used to generate a new revision of the formal specification in step 5).

The objective of this sub-process is threefold: firstly, to create a high conceptual equivalence between the requirement document and the formal specification document, secondly, to identify errors in the specification as early as possible and, finally, to raise the level of comprehensibility of the formal specification.

## B. *Reflection*

Depending on the number of unclear statements and differences between the older an newer revisions of the documents, a second iteration could make sense. The decision about a second (or even third) round might be up to the experts in steps 4) and 5). As quality and complexity measures are used anyway, they could also be used as indicators, telling the experts when to start another cycle and when to proceed without a new translation round.

As can be seen in Fig. 3 (at the top), the sub-process is fitted in the phase of writing down the requirements and producing the (formal) specification. Formal software development models like the Cleanroom process [29], but also agile development models assume at least parts of the requirements document to be available before producing the specification. So, it should be easy to extend existing process models by this sub-process and to gain most benefit out of it.

[1]For Z one could make use of the ViZ environment. ViZ – Visualization of Z Specifications. Project web page at http://viz.uni-klu.ac.at. Page last visited: Feb. 7th, 2013.

It has to be noted that it would also be possible to start with the requirements document as the source document and to take the formal specification as the target document that is produced twice, synthesized, and then translated back to the requirements document. However, as writing a formal specification is time-consuming, producing it twice might be too costly and might slow down, if it is executed twice, the overall development process.

Not addressed in the description of the model so far is the manpower that is needed to conduct all the steps. Measures can very likely be produced automatically, but for the backward translation step(s) at least two translators are needed. And, there might also be the need for a third person that is responsible for preparing the documents for the translators (thus avoiding any bias by pruning away most of the natural language descriptions and implicitly available hints to the original requirements).

The same holds for the translation to another form of representation. Even when large proportions of a specification can be mapped automatically, at least one person is needed for checking for readability and for aspects/concepts of the specification that have been missed.

The last step in the iteration cycle is the activity of requirements consolidation, and here again manpower is needed as at this step most of the stakeholders are brought together.

In all, there is a big overhead that is necessary. Within one iteration at least four new requirement documents and two reports are to be generated by the personnel (I neglect the documents and reports that are generated automatically for a moment). It sounds a lot, but the most important argument for implementing this model is that of the expected raise in quality. Furthermore, there are additional advantages and aspects:

- Several stakeholders are actively involved in the process (especially at step 2a) and step 4)), yielding a higher identification with the project. Agile models already make use of this approach extensively - and the customer is a resource that is usually available "for free".
- Developers, who are later on responsible for producing the code, could be part of the backward transformation team. With this, they can familiarize themselves with the requirements and specification, eventually also speeding up the implementation step.
- Backward transformation is a tedious job, but it is usually faster than generating the formal specification from scratch. And, translators could annotate parts of the specification as "unclear/unable to transform back" ... again speeding up the backward translation process a bit.

So, raising the quality and improving the identification with the specification comes with some cost, but, especially for larger projects and at later stages in the development, the modification of requirements *and* formal specification(s) is for sure extremely costly. Thus, it might pay off and studies taking a closer look at these issues are definitely necessary.

## V. Conclusion and Outlook

This position paper motivates for the perspective that producing and working with formal specifications can be seen as a cross-cultural adaptation problem. It therefore introduces the ideas behind cross-cultural adaptation and, finally, comes up with a model that contains translation (and transformation) loops equivalent to those used in already existing cross-cultural adaptation models.

Formal specifications are still not so widely spread as they should and could be. The objective of this paper is not to present *the* solution to this problem, but it tries to sharpen the understanding of the fact that comprehension issues should not be neglected. By raising comprehensibility a lot of benefits arise: specifications are easier to communicate, they do contain less errors and thus also the implementations (and test cases) are of higher quality.

The price we have to pay is that there is still a lot of work ahead. What is definitely missing is a set of experiments and studies that take a closer look at comprehension aspects. E.g. for object oriented programs we slowly get a feeling of how a "good" program looks like, but what are "good" formal specifications? Are style guides really enough?

Another task, directly related to the model presented in this paper, is to find out which strategy of translation is producing the better results (in respects to quality and resources needed). So, which starting point is better: taking the formal specification as the source document (as is suggested in this paper) or taking the requirements document as the source?

At the moment, a colleague and I are taking a closer look onto the first issue. Based on experiments that will be completed in April 2013, we will try to measure the effects of quality, complexity and comprehensibility of a formal specification onto the number of errors and the time needed for conducting different maintenance tasks.

Formal methods can be integrated into the software engineering community, but removing obstacles is a necessity for it. Taking different cultures and skills into account and raising the comprehensibility by simultaneously demonstrating that not so much changes to one's preferred software development process is needed might be one step towards that direction.

## References

[1] T. Thomas and C. D. Villiers, "Handling diversity in group work in the information systems class," *South African Computer Journal*, pp. 231–236, 2000.

[2] L. Bordyuk, "Linguistic and culture-specific factors for professional success," in *CAD Systems in Microelectronics, 2003. CADSM 2003. Proceedings of the 7th International Conference. The Experience of Designing and Application of*, vol. 26, feb. 2003, pp. 530 – 532.

[3] L. Dubé and G. Paré, "Global virtual teams," *Commun. ACM*, vol. 44, no. 12, pp. 71–73, Dec. 2001. [Online]. Available: http://doi.acm.org/10.1145/501317.501349

[4] D. Anawati and A. Craig, "Behavioral adaptation within cross-cultural virtual teams," *Professional Communication, IEEE Transactions on*, vol. 49, no. 1, pp. 44–56, 2006.

[5] M. Herdman, J. Fox-Rushby, and X. Badia, "A model of equivalence in the cultural adaptation of hrqol instruments: The universalist approach," *Quality of Life Research*, vol. 7, pp. 323–335, 1998. [Online]. Available: http://dx.doi.org/10.1023/A%3A1024985930536

[6] O. Behling and K. S. Law, *Translating Questionnaires and Other Research Instruments – Problems and Solutions*. Sage Publications, Inc., 2000.

[7] M. E. Reichenheim and C. L. Moraes, "Operationalizing the cross-cultural adaptation of epidemiological measurement instruments," *Revista de Saúde Pública*, vol. 41, no. 4, pp. 665–673, 2007.

[8] Y. Y. Kim, *Communication and cross-cultural adaptation: An integrative theory*. Intercommunication series, 2, Clevedon, England: Multilingual Matters, 1988.

[9] M. M. Gordon, "Assimilation in america: Theory and reality," *Daedalus*, vol. 90, no. 2, Ethnic Groups in American Life, pp. 263–285, Spring 1961.

[10] R. E. Park, *Race and Culture*. Glencoe Ill: The Free Press, 1950.

[11] J. R. Baldwin, "Com 372: Intercultural communication," http://my.ilstu.edu/˜jrbaldw/372/Adaptation.htm, Page last visited: Feb. 2013.

[12] D. E. Beaton, C. Bombardier, F. Guillemin, and M. B. Ferraz, "Guidelines for the process of cross-cultural adaptation of self-report measures," *SPINE*, vol. 25, no. 4, pp. 3186–3191, 2000.

[13] F. Guillemin, C. Bombardier, and D. Beaton, "Cross-cultural adaptation of health-related quality of life measures: Literature review and proposed guidelines," *J Clin Epidemiol*, vol. 46, no. 12, pp. 1417–1432, Dec. 1993.

[14] A. Bollin, "Is there evolution before birth? Deterioration effects of formal Z specifications," in *Proceedings of the 13th international conference on Formal methods and software engineering*, ser. ICFEM'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 66–81.

[15] ——, "Metrics for Quantifying Evolutionary Changes in Z Specifications," *To appear in Software Maintenance and Evolution: Research and Practice. Wiley and Sons Ltd.*, 35 pages, Accepted June 2013.

[16] C. Hoare, "An overview of some formal methods for program design," *IEEE Computer*, vol. 20, no. 9, pp. 85–91, 1987.

[17] J. M. Wing, "A specifier's introduction to formal methods," *IEEE Computer*, vol. 23, no. 9, pp. 8–24, 1990.

[18] J. Spivey, *The Z Notation*, ser. C.A.R. Hoare Series, I. Hayes, Ed. Prentice Hall, 1989.

[19] R. Vinter, M. Loomes, and D. Kornbrot, "Applying software metrics to formal specifications: A cognitive approach," in *5th International Symposium on Software Metrics*. Bethesda, Maryland: IEEE Computer Society, 1998, pp. 216–223.

[20] A. von Lamsweerde, *Requirements Engineering From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[21] G. Boolos, "To be is to be a value of a variable (or to be some values of some variables)," *Journal of Philosophy*, vol. 81, pp. 430–450, 1984.

[22] A. Gravell, "What is a good formal specification?" in *5th Annual Z User Meeting*. Springer Verlag, 1990, pp. 137–150.

[23] A. Bollin, "Slice-based Formal Specifiation Measures – Mapping Coupling and Cohesion Measures to Formal Z," in *Proceedings of the Second NASA Formal Methods Symposium*, ser. NASA/CP-2010-216215, C. Muñoz, Ed. NASA, Langley Research Center, April 2010, pp. 24–34.

[24] A. Bollin and A. Tabareh, "Predictive Software Measures based on Z Specifications - A Case Study," in *Submitted to the 2nd Workshop on Formal Methods in the Development of Software, co-located with the 18th International Symposium on Formal Methods, CNAM Paris, France*, 8 pages, August 2012.

[25] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web Services Description Language (WSDL) Version 2.0," http://www.w3.org/TR/wsdl20, 2007.

[26] A. Bollin, "Coupling-based Transformations of Z Specifications into UML Diagrams," *International NASA Journal on Innovations in Systems and Software Engineering*, vol. 7, no. 4, pp. 283–292, 2011.

[27] D. Jackson, *Software Abstractions - Logic, Language, and Analysis*. The MIT Press, Cambridge, Massachusetts, 1996.

[28] C. B. Jones, *Systematic Software Development Using VDM*, 2nd ed. Prentice Hall International, 1990.

[29] H. Mills, M. Dyer, and R. Linger, "Cleanroom software engineering," *IEEE Software*, vol. 4, no. 5, pp. 19–25, 1987.