Journal of Universal Computer Science, vol. 7, no. 8 (2001), 692-709
submitted: 18/5/01, accepted: 21/8/01, appeared: 28/8/01 © Springer Pub. Co.

Ten Years of Historical Development "Bootstrapping" VDMTools®

Peter Gorm Larsen
(IFAD A/S
Forskerparken 10A
DK-5230 Odense M, Denmark
peter@ifad.dk)

Abstract: This article provides a historical overview of a decade of the development of the IFAD VDM Toolboxes commonly referred to as **VDMTools®**. All along, the existing tools have been used in the development of new major components. This kind of "bootstrapping" approach where a CASE tool is developed by taking "its own medicine" is seldom used. However, we believe that this approach is important to be able to better understand what the most important improvements are for the users in practice. This article also describes how the different components have been maintained by a changing development team. We feel that the decisions we have made regarding the parts of the tool which have been formally specified and the parts which have been developed conventionally may provide valuable input for others considering the use of formal specification. The overall organisation of the development environment may also be interesting for other developers.
Key Words: Formal methods, Software engineering, VDM, Tool support
Category: D.2.1

1 Introduction

The Vienna Development Method (VDM) [BJ82, Luc87, Jon90] has a specification language (VDM-SL) which is standardised by the International Organisation for Standardisation (ISO) [PL92, ISO96]. VDM is one of the most mature formal methods, primarily intended for formal specification and subsequent development of functional aspects of software systems. Its specification language VDM-SL [Daw91, FL98] is used during the specification and design phases of a software development project, and it supports the production of correct high quality software. An extension of this notation is supported by the IFAD VDM-SL Toolbox. Another object-oriented extension called VDM++ [Gro00a], is supported by the IFAD VDM++ Toolbox. Commonly these two tools with their different features are referred to as **VDMTools®**.

The development of the first component of the IFAD VDM-SL Toolbox [ELL94, Muk95] started in 1990 in an ESPRIT project called IPTES [Edi93]. Among other tasks in this project IFAD was responsible for developing an interpreter for an executable subset of VDM-SL in a larger tool dedicated to interpreting SA/RT [HP87] specifications in an incremental and heterogeneous manner. A part of a type checker (also called the static semantics) for this subset and a part of a C++ code generator from this subset and for VDM++ was produced in the ESPRIT projects IDERS [ABCH95] and Afrodite [Lan95]. In the INFORMA [INF] project the VDM++ notation was improved, and a Java code generator and an interpreter for VDM++ were developed. In the TIAPS [AF97a, AF97b] and PROSPER projects proof support for VDM-SL was developed. In the SPECTRUM project [ALR98] a feasibility study of stepwise development from VDM to the B notation [Abr96] and then carrying out proofs there was conducted.

In the ESSI project ConForm [FLBG95, LFB96] British Aerospace evaluated the effect of using the formal specification notation VDM-SL and the IFAD VDM-SL Toolbox in the development of a trusted gateway. Similar experiments have been made in the ESSI projects PICGAL [DLV97a, DLV97b] and ISEPUMS [PT99, Puc00] where the C++ code generator was also evaluated. In the trial project UseGat [Kir97] a closer integration with SA/RT was made using the "Software through Pictures" product [IDE94]. Finally the trial project VICE [MBD+00] extended the VDM++ technology with more support in the real-time area. Since 1994 **VDMTools**® has been sold commercially all over the world and different users have also published papers about some of the work they have conducted with it [SL99, vdBVW99b, vdBVW99a, HA00, FYH96]. In addition, some reports of applications can be found in the Toolbox Newsletters [New]. Since the focus of this article is the development of **VDMTools**® instead of its use outside IFAD we will not go further into these in this article.

This article first provides an overview of the different components of **VDM-Tools**®. This is followed by a historical overview where the varying size of the development team can be seen in Section 3. A short overview of the different aspects of the software engineering environment is given in Section 4. The actual bootstrapping process is then presented in Section 5. After this section, the development and maintenance approach for three different categories of components is discussed. In Section 9 the major development decisions are considered in retrospect with a critical eye. Finally, a few concluding remarks are given.

2 VDMTools Overview

An overview of the connection between the components in **VDMTools**® is given in Figure 1 (note that the dashed lines here indicate that this component is under development). The components can be classified into three categories:

- Components developed conventionally, i.e. with an informal textual description as requirements and documentation and then the actual code with appropriate comments.
- Components formally specified in VDM-SL and hand implemented afterwards. For these components the documentation is a mixture of formal text

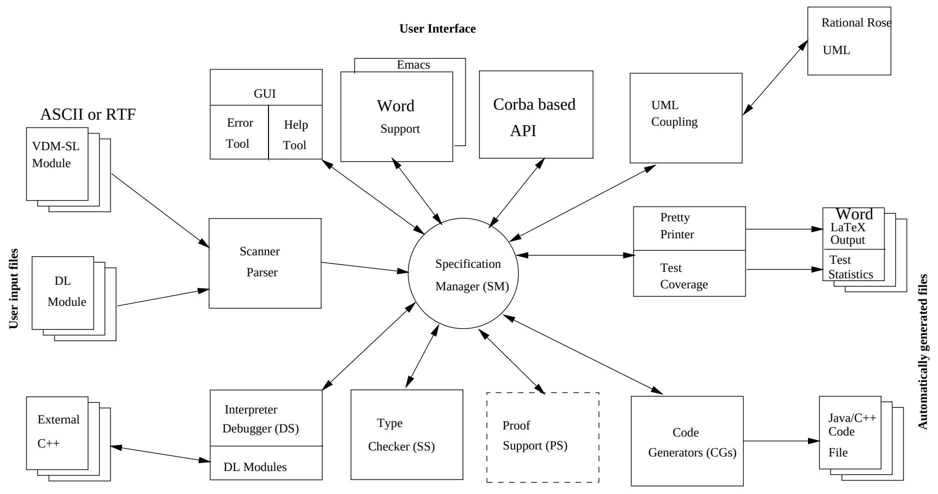

Figure 1: Overview of VDMTools

and explanatory annotations whereas the code follows certain coding standards which makes it easy to find the code corresponding to a specific part of a specification.

– Components specified in VDM-SL or VDM++ and automatically code generated. For these components the documentation at the VDM level is actually used as source text for the corresponding code as well. In effect the VDM notations are here used as high-level programming languages.

The category for a component is determined both from the nature of the component and the historical background for it in the bootstrapping process. In general it can be said that the strategy is to use formal specification whenever we find it beneficial. In particular in cases where complex functionality or a complex data structure is involved, formal specification would be used. We will return to this in Section 5.

Below we present three subsections for the different categories where we give a short explanation of each of the components (abbreviations used for the components later in this article are given in brackets after the name of the component):

2.1 Components Developed Conventionally

Scanner/Parser: This have been developed conventionally using parser generator technology (`flex` and `bison`). This component is responsible for checking the syntax of specification files (a mixture of informal text and formal definitions is allowed here) developed by the user. If the syntax is correct an abstract syntax representation of the specification is produced and given to the specification manager. Otherwise a list of errors is produced. Here it may also be worth noting that a non-standard intelligent error recovery and error reporting approach has been used to replace the standard way for error reporting in such a parser generator [Nie95].

Pretty Printer: This has been developed conventionally directly in C++. This is a functionally simple component which is able to produce a pretty printed version of a specification either using a set of LaTeX macros produced for VDM-SL at NPL [DL95] or in Rich Text Format (RTF) which can be used directly inside the Microsoft Word editor. This component is also able to merge the informal source text surrounding the formal definitions in the files provided by the user. If the interpreter has been used with a special option for collecting test coverage information the pretty printer will also be able to display this information.

Interfaces: Three different interfaces for the Toolbox exists: a CORBA based interface enabling external access to **VDMTools**® functionality, a graphical user interface and a command-line interface. All of these have been developed conventionally using respectively OmniORB [Lo] and Tcl/Tk [Ous94].

Note that all of the components which have been developed conventionally are interfaces to **VDMTools**® from users point of view. Such components are normally developed conventionally. This is typical in the sense that these are either front-ends or back-ends and in most other applications we have been involved with applying VDM such components are normally developed conventionally.

2.2 Components Formally Specified and Manually Implemented

Interpreter/Debugger (Dynamic Semantics, DS): This has been specified in VDM-SL and hand coded afterwards. This component is able to interpret an expression using definitions provided by the user in the modules. As in a programming environment there is debugging functionality which enables the user to set breakpoints etc. Initially this specification document was between 60 and 70 pages [LL91]. Two major extensions to the first subset (modules and Dynamic Link (DL) modules) increased the size of the specification of this component to slightly above 200 pages including comments explaining the actual formal definitions and cross reference indices. The interpreter/debugger component has been extended with a Dynamic Link (DL) feature which enables the user to interpret a system where some modules are specified while others only are only present in C++ and then dynamically linked together with the Toolbox process [FL96].

The initial version of this dynamic semantics worked directly as an abstract syntax tree traversal. During 1999 this was totally redeveloped using a stack-machine approach enabling up/down functionality in the debugger and handling of multiple threads in the VDM++ interpreter. Extensions of the VDM-SL interpreter with support for implicit definitions have also been defined [Frö98] but these have not yet been incorporated in **VDMTools**®.

Type Checker (Static Semantics, SS): This has been specified in VDM-SL and hand coded afterwards. This component is able to carry out static checks of the internal correctness of the specifications provided by the user. This can be done in two modes; one for checking a specification for being "possibly correct" (this is close to the level of type checking carried out by a compiler for a programming language); and a mode for checking a specification for being "definitely correct". The latter mode corresponds to producing errors for all the places where "run-time errors" *could* occur. It is the intention that the "definite" mode should be changed to produce proof obligations instead of regular errors [Aic97, AL97]. This is closely related to the proof support currently under development. The specification here is approximately 350 pages of VDM-SL organised in 13 modules.

Code Generators (CG): These have been specified in VDM-SL and hand coded afterwards. This component is able to produce fully executable C++/Java code using a VDM C++/Java library providing generic implementations of VDM concepts (the C++ version of this library is also used internally in the other Toolbox components). For constructs which are not covered by the subset of VDM for which C++ code can be automatically produced, there are a number of ways the user can interface to the code generated for the remaining part of the specification. This specification is more than 500 pages including comments explaining the formal definition and illustrating the code being generated. The Java code generator is even able to take the concurrent part (including the synchronisation constraints) of VDM++ into account [Opp99].

All of these components are complex kernel functionality and it is not accidential that such components are formally specified due to their importance and complexity.

2.3 Components Formally Specified and Code Generated

Specification Manager (SM): This has been specified in VDM++ and UML [Gra97] (originally OMT [RBP+91] was used) and automatically code generated. The features for combining manually implemented code with automatically produced code are used here. This component is responsible for keeping track of the status of all modules/classes which have been syntactically correct provided by the user. The component was developed using design patterns [ERRJ95]. The specification here is approximately 4000 lines of VDM++ organised in 28 classes.

Proof Support (PS): The proof obligation part is largely automatically code generated whereas the proof support is made conventionally on top of the HOL98 system conventionally. In the TIAPS project proof support was attempted using Isabelle [Pau94] and a few experiments with PVS [ORSV95]. Using Isabelle we had problems with the level of automation which was possible to achieve whereas using PVS we had problems with the level of control over the automation. In the PROSPER project [DCN$^+$00] this work was redone using HOL98 [Gor87, NS00] to get a higher degree of automation and integration. This component contains a proof obligation generator, automatic proof support and interactive proof support. Many of the proof obligations generated can be automatically proved but some of them require interactive proofs to be made by the user. This component includes 5 parts which have been specified in VDM-SL with approximately 28000 lines. In addition there are around 13000 lines of ML (for HOL theories) and 18000 lines of Java (for the user interface).

Coupling to graphical tools: In the UseGat project, back in 1995, a coupling of **VDMTools**® to the commercial “Software through Pictures” tool has been instantiated with SA/RT. This enabled a graphical animation directly on top of data flow diagrams but it turned out to be too inefficient and the focus on object orientation meant fewer potential users of SA/RT. Consequently a coupling to Verilog’s LOV/OMT tool was made to VDM++ instead. This was later replaced with a coupling to Rational Rose [Ros] and UML when Verilog was taken over and the advent of UML meant that OMT was obsolete. The specification of the mapper between VDM++ and UML is approximately 1500 lines of VDM++.

3 The Historical Overview

A rough overview of the staff allocation to the development of a major part of **VDMTools**® components can be seen in Figure 2. The initials for the permanent staff members are listed in the left-hand side of the figure. Time is shown along the horizontal axis and the bars for each of the employees indicate the period where they have been involved partly with **VDMTools**® development. This does not mean that the employee has been working full time on **VDMTools**® development or maintenance, but simply that a part of that employees

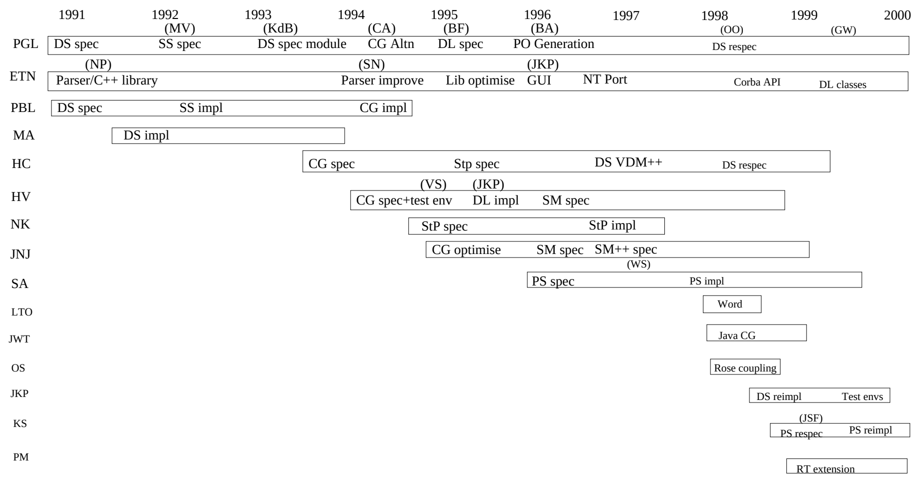

Figure 2: Historical Overview of the Staff Allocation

time has been allocated to **VDMTools**® matters in the given period. On the bars the parts which an employee have been involved with is identified. In the cases where a visitor has been involved with the development the initials are put in brackets next to the bar. These visitors have either been external experts we have purchased to assist us, or students.

The students have either carried out feasibility studies or developed the initial version of a component in collaboration with the permanent staff member to which the student was allocated. The students have all spent between 1 and 10 months at IFAD, mostly as their MSc thesis project. The students we have had came from The Netherlands, Germany, Austria, France, USA and Denmark. In general we have been very satisfied with the students we have had and in all cases, where they have developed a VDM specification of a part of a Toolbox component, we have been able to take over and maintain their specifications without major problems. However, it has been much more difficult to take over the code developed by students. Thus, in the future we envisage simply letting the students specify their work in VDM-SL or VDM++ and then we will develop the necessary code or use automatic code generation. Even though one needs to invest time in supervising the students we feel that students are able to present their ideas at the specification level in this way, it is a good value for money.

4 Development Environment

The fact that we believe in the use of VDM industrially does not mean that we also have neglected other software engineering aspects. It is essential to remember that VDM is just one tool to be used when appropriate and that a proper development environment for the informal parts of software engineering is also essential. In this section we provide an overview of the most important principles underlying our development environment in the Toolbox development team.

In all source files (both test cases, test environments, specifications and code) `ifdef` directives are used to take care of the differences between VDM-SL and VDM++. All files are then pre-processed before they are given to **VDMTools**® and the C++ compilers respectively. For the implementation `ifdef` directives are also needed to cope with differences between the different platforms.

All source files are maintained under the version control system CVS [CVS] which enables different developers to work in parallel on the same components and merge their changes together when it is appropriate. Whenever a change is made in a VDM specification which is hand-coded the `ediff` facility inside the `Emacs` editor is used to see the exact differences between two versions of a source file. This is an advanced diff tool which can be operated conveniently to identify precisely the changes between the newest version of the specification and the revision of it which was last implemented.

For each of the components a test environment has been set up which is used for regression testing of any changes made to it. New test cases are introduced whenever new features are introduced. The test environments automate as much as possible of the testing of the correct behaviour of **VDMTools**®. Thus, for each test case there is a corresponding expected result. For the components specified using VDM, test coverage is also automatically produced and the entire specification of the component can be pretty-printed using **VDMTools**® and LaTeX.

Over the years the development team developed a number of procedures which are followed in the development and maintenance of **VDMTools**® [Gro01a]. This includes a task catalogue identifying all the ideas for improvements which have arisen internally or externally [Gro01b]. For each task a description and an estimate is made and every developer follows procedures in carrying out tasks. There are also special procedures connected to official releases of **VDMTools**®.

5 The Bootstrapping Process

In Figure 3 it is shown how the different specifications of components which have been made are used in the development of other components which have been

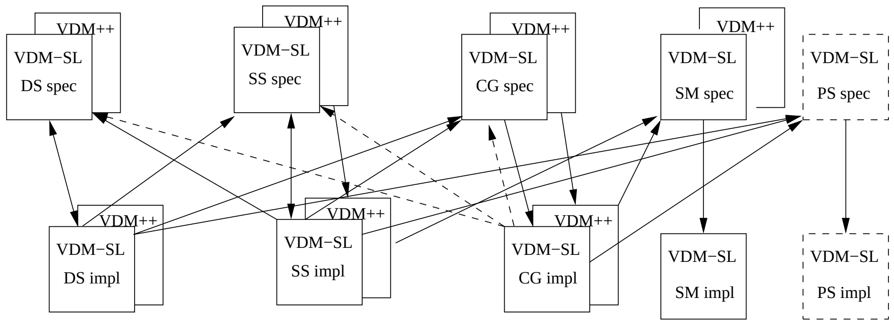

Figure 3: Overview of the Bootstrapping Process

specified. This bootstrapping process is expected to continue in future development. At the top of the figure the specified components are placed, whereas the corresponding implementations are placed at the bottom. The arrows between these components indicate how a component has been used in the development and/or maintenance of other components.

Each of the specifications which has been hand-coded (interpreter, type checker and code generator) has naturally been used as basis for the corresponding implementations. The implementation of the interpreter and the type checker have then been used to test and improve all of the components which have been specified in VDM. They have also been used on their own specifications! Note that this kind of bootstrapping approach is often used for compilers for programming languages, but we are not aware of any other tools supporting specifications which have taken this approach seriously. The dashed arrows from the code generator implementation to the specifications of the components which has been hand-coded are used to identify that we have tried to use the code generator on these components. However, we have decided to keep the hand-coded components so far because of efficiency reasons, but it is possible that we would use the code generator for these components some time in the future if the automatically generated code becomes sufficiently efficient. Notice that then we would also be able to use the code generator implementation on its own specification. The solid arrows from the code generator implementation indicates that the code generator has been used to automatically generate the implementation of a component and the generated code is used in the **VDMTools**® products. The overall strategy is to improve the code generator such that we will be able to use it directly on all new components we formally specify in the future.

6 Purely Hand-Coded Components

Despite the fact that we "take our own medicine" we do not always use formal specification. This was one of the myths about promoters of formal methods refuted in [BH95] and we cannot agree more. We take a pragmatic approach where we consider for each component whether there is any benefit from formally specifying parts of it. For three of the components used in the Toolboxes we have not found that it was worth making a formal specification. These are the scanner/parser component, the pretty-printer component and the user interface component.

1. For the scanner/parser component we feel that existing parser generators are very convenient for solving this kind of problems. We feel that formal specification would not be able to improve the development of such a component.
2. For the pretty-printer component we feel that it has a relatively simple functionality; an abstract syntax tree traversal is required. Since this component is also not considered to be very critical we feel that a formal specification of such a component would be rather low-level and it would not be able to improve the quality of the component.
3. For the user interfaces we believe that a formal specification would not be worthwhile. The point here is that existing GUI tools are very fast to use, and from prototypes of such user interfaces one can get more valuable feedback than from a formal specification. However, more recently the more complex parts of the GUI logic have been specified in VDM++ and automatically code generated.

7 Formally Specified and Hand-Coded Components

For the components where we have considered the use of formal specification valuable we have used one of the VDM notations (VDM-SL and VDM++). Since these components are developed in a bootstrapping fashion it was initially necessary to hand-code each of them. It is interesting to note here that the structure of the specifications in general is very close to the structure of the corresponding code. Thus, we have a number of procedures which makes it easier to maintain both the specification and the code. As an example of this there are naming conventions which make it easy to find a function from the specification in the corresponding implementation.

The first version of the VDM-SL Interpreter (which was the first component to be specified) was initially specified in VDM-SL itself directly using the NPL

LaTeX macros for VDM-SL [DL95]. This first large specification was not written to be executable. However, it turned out that we only found it useful to use purely implicit constructs in a few places. A more explicit version of these functions/operations was made enabling us to interpret the specification of the interpreter by itself. All the specifications we have developed are written in the executable subset of the VDM notations and we are hereby able to interpret test arguments at the VDM level. In addition the specifications have been used as a communication medium between the members of the development team. Note that by using invariants and pre-conditions we have had an advantage of using VDM at the design exploration stage. This has enabled us to discover inconsistent use of data structures and functionality at a very early stage.

In the maintenance phase of the formally specified components errors reported are first reproduced at the specification level. After the problem has been fixed at the specification level the entire test environment for that component is run at the specification level before it is checked into the revision control system. The implementation of the changes are then done subsequently (often by another person) using Ediff.

In the development of these components we have gathered a few measures. In the process of going from a specification to an implementation in average between 30 and 50 lines of specification (including comments) are implemented per hour. Only around a 4th of the total development time is used for implementation and test of implementation (reusing test cases from the specification level). Thus, around 75 percent of the effort is spent on specifying the components and setting up a test environment for testing at the specification level.

For the first specifications we made we did not have any precise measurements of the effort spend on producing the actual specifications. However, for the VDM++ interpreter 12.7 lines of specification have been written per hour. This figure includes understanding the extension of VDM-SL incorporated in VDM++ and devising and running tests. This figure is quite similar to industrial standards for most programming languages but because the notation is more abstract we are able to formulate more functionality than if for example C++ was used directly.

At an earlier stage of the development of the **VDMTools**® code generators a few metrics were gathered. These can be seen from Figure 4. The productivity for this part is slightly less than indicated above and the proportion of time between specification and implementation is not as high as above. We believe that this is caused partly because of the complexity of this component and partly because new employees were involved in the development here.

VDM-C++ code generator	LOC	hours	loc/hour
VDM document	21244	1090	
Raw VDM spec	7081	1090	6.5
C++ coding	23322	789	29.6
In total	23322	1879	12.4

Figure 4: Overview of VDMTools

8 Specified and Code Generated Components

After the development of the C++ code generator we have had to consider whether components which have been formally specified should be code generated or hand-coded. Naturally there is a trade-off between cost-effectiveness and efficiency of the final production code. In general the strategy is to use automatic code generation for new components if the efficiency of the generated code is reasonable. If improvements of the C++ code generator are necessary we would then take that investment into the consideration, but the improvement would also be of value for the customers.

In general the components which are specified in VDM and automatically code generated are tested in the same way as described above for the components which are specified and hand-implemented. However, here we do not need to rerun the test cases with the final code.

9 Looking in Retrospect

When one looks back at a development of a software system over a decade there are probably always some choices one would have taken differently. In this section we try to provide insight into the author's personal feelings for the most important decisions where different choices could have been better with the knowledge of today.

As it could be seen from the references in the introduction of this article a significant portion of the funding for the development in **VDMTools**® came from the CEU R&D programmes (in particular ESPRIT). Without that support the development of **VDMTools**® would clearly never have come as far as it has. However, such projects also have a large administrative overhead and the development taking place inside such projects would normally not be the most important ones from a commercial point of view. Thus, IFAD's own investment has also been significant taking its size into account. The combination of these two forms for funding the development has been much slower than we would have liked.

On the more technical side the overall architecture has not been well enough designed, because of the evolution of **VDMTools**® during this decade. The development team has focused on the specification and design of the different components without always considering the overall connection between them in enough detail. It was not felt that it was necessary to specify how the different components depended on each other and this has been a mistake. This has resulted in different kinds of unnecessary internal transformations. In addition, it is impossible to compile the different components separately. This can be blamed both on our lack of understanding for the need of focussing on this, but also on the funding approach in a series of EU supported projects (each with its own focus). A new effort to have a more service based architecture has been initiated, and it is believed that this will make the different components less dependent upon each other.

On the graphical user interface side it can be said that we did not pay enough attention to its importance and made several mistakes [Joh00]. This is currently being redone, entirely replacing the Tck/Tk interface using Qt instead [Qt00]. With our focus on formal specifications of the different kernel components we must admit that we underestimated the importance of this part.

There has been a couple of the specifications of components which have been redeveloped during this decade and naturally one can argue that it would have been better to develop it right the first time. However, here we would claim that this would be virtually impossible because the reasons for redeveloping them came from new desires, unknown when the specifications where first developed.

If we had the chance to turn back the clock we would probably have opened **VDMTools**® more for add-ins and have enhanced the academic collaboration at a much earlier stage. We feel that the use of abstract models formulated in VDM is best spread if students learn about them already at university. Thus, we have now made **VDMTools**® freely available for academic purposes for universities. In addition there is now an API to **VDMTools**® [Gro00b] which enables more "power users" to experiment with their own extensions. More possibilities for adding external features are currently under development.

Finally, we would like to state that all in all we feel that looking in retrospect we would develop most of **VDMTools**® in the same way as we have done this decade with the few exceptions mentioned above.

10 Concluding Remarks

Many things change during a decade. As explained above some development choices taken must be reconsidered later on. One must also be ready to throw away features which did not work as intended or did not have as many potential users as originally estimated, e.g. the SA/RT combination.

Staff for developing products such as this are bound to change significantly over time. This naturally influences the way in which software must be developed and maintained. A high level of documentation is required. As most development teams we have certainly also run into situations where we have not documented parts sufficiently. However, we firmly believe that because the most complex features have been described using VDM, we have been able to maintain the product over a decade with a changing development team over its life time.

When we first started the development of **VDMTools**® we did not envisage that we would ever come as far as we have come today. However, now we have new visions for how it should be improved in the future. The most surprising experience during this decade has been how much effort it takes to go from a proof of concept prototype to a commercial product. To move out of the "academic-nature" arena one needs to support ever changing platforms (UNIX, Linux, Windows 95, 98, 2000, NT) and to interoperate with the de facto industrial standard tools such as Microsoft Word and Rational Rose. We have done that, but this kind of integration takes more time than one would imagine. In addition the industrial requirements for robustness of such a product are much higher than known from the academic "proof of concept" prototypes. Furthermore the target for such a product is continously changing, for example with the introduction of Java.

We think that the pragmatic approach we have had to our development taking ones own "medicine" on selected components has been beneficial. Using VDM has certainly helped us to master the complexity of our own systems. Finally, I also think that it is important that it has been **fun** developing in this way! We find it very important that everyone involved in the development of a product such as this firmly believes in the benefits in using it.

Acknowledgements

First of all I would like to thank the IBM Vienna group for inventing VDM and in particular Dines Bjørner for introducing it to me as enthusiasticly as he did. My thanks also go to all the customers, employees and students which I have worked with during the last 10 years around the production of professional tool support and services for the VDM notations. In particular I would like to thank Andreas Kerschbaumer, Paul Mukherjee and Kim Sunesen for proof reading this paper. Finally, I would like to thank the different funding organisations which have made this development possible.

References

[ABCH95] Alejandro Alonso, Luciano Baresi, Hanne Christensen, and Marko Heikkinen. IDERS: An integrated environment for the development of hard real-time systems. In *EUROMICRO Real-Time Workshop*. IEEE, June 1995.

[Abr96] J.-R. Abrial. *The B Book – Assigning Programs to Meanings*. Cambridge University Press, August 1996.

[AF97a] S. Agerholm and J. Frost. An Isabelle-based theorem prover for VDM-SL. In *Proceedings of the 10th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'97)*, LNCS. Springer-Verlag, August 1997. Also available as technical report IT-TR: 1997-009 from the Department of Information Technology at the Technical University of Denmark.

[AF97b] Sten Agerholm and Jacob Frost. Towards an integrated CASE and theorem proving tool for VDM-SL. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, September 1997. ISBN 3-540-63533-5.

[Aic97] Bernhard Aichernig. A Proof Obligation Generator for the IFAD VDM-SL Toolbox. Master's thesis, Technical University Graz, Austria, March 1997.

[AL97] Bernhard K. Aichernig and Peter Gorm Larsen. A proof obligation generator for VDM-SL. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 338–357. Springer-Verlag, September 1997. ISBN 3-540-63533-5.

[ALR98] Sten Agerholm, Pierre-Jean Lecoeur, and Etienne Reichert. Formal Specification and Validation at Work: A Case Study using VDM-SL. In *Proceedings of Second Workshop on Formal Methods in Software Practice*. ACM, Florida, March 1998.

[BH95] Jonathan P. Bowen and Michael G. Hinchey. Seven more myths of formal methods. *IEEE Software*, 12(3):34–41, July 1995.

[BJ82] D. Bjørner and C.B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall International, 1982.

[CVS] Concurrent Versions System. http://www.cvshome.org/.

[Daw91] John Dawes. *The VDM-SL Reference Guide*. Pitman, 1991. ISBN 0-273-03151-1.

[DCN$^+$00] Louise A. Dennis, Graham Collins, Michael Norrish, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom Melham. The PROSPER Toolkit. In *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Germany, March/April 2000. Springer-Verlag, Lecture Notes in Computer Science volume 1785.

[DL95] I.P. Dickinson and K.J. Lines. Typesetting VDM-SL with VDM-SL macros. Technical report, National Physical Laboratory, Teddington, Middelsex, TW11 0LW, UK, July 1995.

[DLV97a] Lionel Devauchelle, Peter Gorm Larsen, and Henrik Voss. PICGAL: Lessons Learnt from a Practical Use of Formal Specification to Develop a High Reliability Software. In *DASIA'97*. ESA, May 1997.

[DLV97b] Lionel Devauchelle, Peter Gorm Larsen, and Henrik Voss. PICGAL: Practical use of formal specification to develop a complex critical system. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 221–236. Springer-Verlag, September 1997. ISBN 3-540-63533-5.

[Edi93] Sandro Bologna (Guest Editor). Special Issue: Incremental Prototyping of Real-Time Systems. *Real-Time Systems*, 5(2/3), May 1993. 7 papers on the methodological and tool support developed in the ESPRIT II IPTES EP5570 project.

[ELL94] René Elmstrøm, Peter Gorm Larsen, and Poul Bøgh Lassen. The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications. *ACM Sigplan Notices*, 29(9):77–80, September 1994.

[ERRJ95] E.Gamma, R.Helm, R.Johnson, and J.Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, 1995.

[FL96] Brigitte Fröhlich and Peter Gorm Larsen. Combining VDM-SL Specifications with C++ Code. In Marie-Claude Gaudel and Jim Woodcock, editors, *FME'96: Industrial Benefit and Advances in Formal Methods*, pages 179–194. Springer-Verlag, March 1996.

[FL98] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development.* Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[FLBG95] John Fitzgerald, Peter Gorm Larsen, Tom Brookes, and Mike Green. *Applications of Formal Methods*, chapter 14. Developing a Security-critical System using Formal and Convential Methods, pages 333–356. Prentice-Hall International Series in Computer Science, 1995.

[Frö98] Brigitte Fröhlich. *Towards Executability of Implicit Definitions.* PhD thesis, TU Graz, Institute of Software Technology, September 1998.

[FYH96] Mitsuyoshi Fukuda and Takahiko Ogino Yuji Hirao. VDM Specification of an Interlocking System and a Simulator for its Validation. In *WCRR(World Conference on Railway Research)*, Colorado Springs, USA, 1996.

[Gor87] M. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis.* Kluwer Academic Publishers, 1987.

[Gra97] Grady Booch and Ivar Jacobson and Jim Rumbaugh. The Unified Modelling Language, version 1.1. Technical report, Rational Software Corporation, September 1997. Available at: http://www.rational.com/.

[Gro00a] The VDM Tool Group. The IFAD VDM++ Language. Technical report, IFAD, October 2000. ftp://ftp.ifad.dk /pub/vdmtools/doc/langmanpp_letter.pdf.

[Gro00b] The VDM Tool Group. VDM Toolbox API. Technical report, IFAD, October 2000.

[Gro01a] IFAD VDM Toolbox Group. Quality Plan for the VDM-SL and VDM++ Toolboxes. Technical report, IFAD, Forskerparken 10, 5230 Odense M, Denmark, January 2001.

[Gro01b] IFAD VDM Toolbox Group. Task Catalogue for the VDM-SL and VDM++ Products. Technical report, IFAD, Forskerparken 10, 5230 Odense M, Denmark, January 2001.

[HA00] Johann Hörl and Bernhard K. Aichernig. Validating voice communication requirements using lightweight formal methods. *IEEE Software*, May 2000.

[HP87] D.J. Hatley and I.A. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House, New York, 1987.

[IDE94] IDE. *Software through Pictures/ Core. Fundamentals of StP*, February 1994. Release 2.

[INF] INFORMA — integrated formal approaches for embedded real-time systems. http://www.ifad.dk/Projects/informa.htm.

[ISO96] Peter Gorm Larsen and Bo Stig Hansen and Hans Brunn and Nico Plat and Hans Toetenel and Derek Andrews and John Dawes and Graham Parkin and others. Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language — Part 1: Base language, December 1996.

[Joh00] Jeff Johnson. *GUI Bloopers*. Academic Press, 2000.

[Jon90] Cliff B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall International, Englewood Cliffs, New Jersey, second edition, 1990. ISBN 0-13-880733-7.

[Kir97] UseGat Consortium. Editor: Niels K. Kirkegaard. Use of integrated graphical & textual formal specification languages in industry — final report. Technical report, The Institute of Applied Computer Science (IFAD), September 1997.
UseGat Doc.id.: USEGAT-IFAD-52-V2.1.

[Lan95] Kevin Lano. Applications of formal methods to safety-critical transport systems: the afrodite project. *Safety Systems - The Safety-Critical Systems Club Newsletter*, 4(2):10–12, January 1995.

[LFB96] Peter Gorm Larsen, John Fitzgerald, and Tom Brookes. Applying Formal Specification in Industry. *IEEE Software*, 13(3):48–56, May 1996.

[LL91] Peter Gorm Larsen and Poul Bøgh Lassen. An Executable Subset of Meta-IV with Loose Specification. In *VDM '91: Formal Software Development Methods*. VDM Europe, Springer-Verlag, March 1991.

[Lo] Sai-Lai Lo. *The omniORB2 version 2.5 User's Guide*. Olivetti and Oracle Research Laboratory.

[Luc87] Peter Lucas. VDM: Origins, Hopes, and Achievements. In Airchinnigh Bjørner, Jones and Neuhold, editors, *VDM '87 VDM - A Formal Method at Work*, pages 1–18. VDM-Europe, Springer-Verlag LNCS 252, 1987.

[MBD$^+$00] Paul Mukherjee, Fabien Bousquet, Jerome Delabre, Stephen Paynter, and Peter Gorm Larsen. Exploring Timing Properties Using VDM++ on an Industrial Application. In Juan Bicarregui and John Fitzgerald, editors, *The Second VDM Workshop*, September 2000.

[Muk95] Paul Mukherjee. Computer-aided Validation of Formal Specifications. *Software Engineering Journal*, pages 133–140, July 1995.

[New] The Toolbox Newsletter. 5 issues between 1995 and 2000.

[Nie95] Søren Nielsen. Error messages and error recovery for VDM-SL. Master's thesis, Odense University, Department of Mathematics and Computer Science, March 1995.

[NS00] Michael Norrish and Konrad Slind. A Thread of HOL Development. *The Computer Journal*, 2000. To appear in a special issue in honour of Graham Birtwistle.

[Opp99] Oliver Oppitz. Concurrency extensions for the VDM++ to Java code generator of the IFAD VDM++ toolbox. Master's thesis, TU Graz, Austria, April 1999.

[ORSV95] S. Owre, J. Rushby, N. Shankar, and F. VonHenke. Formal Verification of Fault-Tolerant Architectures: Prolegomena to the Design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley Publishing Company, Inc., 1994.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover.* Springer-Verlag LNCS 828, 1994.

[PL92] Nico Plat and Peter Gorm Larsen. An overview of the ISO/VDM-SL standard. *Sigplan Notices*, 27(8):76–82, August 1992.

[PT99] Armand Puccetti and Jean Yves Tixadou. Application of VDM-SL to the Development of the SPOT4 Programming Messages Generator. In John Fitzgerald and Peter Gorm Larsen, editors, *VDM in Practice*, pages 127–137, September 1999.

[Puc00] Armand Puccetti. Improving the Software Evolution Process Using Mixed Specification techniques (ISEPUMS – Final Report. Technical Report ESSI Project 27492, CS-SI, March 2000.

[Qt00] Qt, December 2000. http://www.trolltech.com/.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall International, 1991. ISBN 0-13-630054-5.

[Ros] Rational Software Corporation, `http://www.rational.com/rose`. *Rational Rose 2000 Using Rose.*

[SL99] Paul R. Smith and Peter Gorm Larsen. Applications of VDM in Banknote Processing. In John Fitzgerald and Peter Gorm Larsen, editors, *VDM in Practice!*, pages 67–79, September 1999.

[vdBVW99a] Manual van den Berg, Marcel Verhoef, and Mark Wigmans. Formal Specification and Development of a Mission Critical Data Handling Subsystem, an Industrial Usage Report. In John Fitzgerald and Peter Gorm Larsen, editors, *VDM in Practice*, pages 95–98, September 1999.

[vdBVW99b] Manual van den Berg, Marcel Verhoef, and Mark Wigmans. Formal Specification of an Auctioning System Using VDM++ and UML, an Industrial Usage Report. In John Fitzgerald and Peter Gorm Larsen, editors, *VDM in Practice*, pages 85–93, September 1999.