A formally proven data structures library intended for Formal Methods teaching

M.Sc. Dissertation

Alexis Chevalier
M.Sc. Student in Software Engineering
Oxford Brookes University

Supervisor: Ian Bayley

***Abstract*—Formal methods are mathematically based validation techniques used in software engineering to prove the validity of a given system towards a specification. In this project we created a data structure library indented for educational purposes using Formal Methods as the main validation system. The library will be used in the context of a module of Formal Software Engineering taught at Oxford Brookes University to M.Sc. Students to help students to have a better understanding of Formal Methods with formally proven data structures and algorithms. In this paper we will summarize our work on the project, introduce our main choices and research, detail our implementation process and present our results before concluding with a critical discussion of our work along with some recommendations for future work.**

***Keywords*—Formal Methods, Dafny, Education, Data Structures.**

I. Introduction

Formal Methods are a set of validation techniques based on mathematics used to validate the correctness of a given system towards a specification, in this dissertation we studied those methods in the context of Software Engineering, where they are used to prove the validity of a software system. Oxford Brookes University proposes a module of Formal Software Engineering [1] to M.Sc. students in Computer Science and Software Engineering, a part of this module is taught using Spec# as specification language in order to show a practical approach to the students using a language similar to C#. In this project we will design, specify and implement a data structure library using a specification language (Spec# or similar). This library is indented to be used as a teaching material for the previously mentioned course in order for the students to have access to a large set of examples of systems validated using Formal Methods. Our main goal is to improve the understanding of Formal Methods among the students through helpful examples picturing various validation techniques and ultimately increase the interest towards the field.

During our research, we didn't find any clues about an existing formally proven library, which is unfortunate since Formal Methods could ensure the validity of a given library towards its specification and would provide and additional level of safety when using the library in another project. We believe that the originality of our project resides in the concept of providing a formally proven library, even if it is not complete, it could be a start and encourage the completion of this project or the creation of related projects. Also, we want this library to be a useful teaching material which could be used to increase the interest towards Formal Methods since the current presence of the field in the education is not very bright as explained in section II-B.

In this paper we will summarize the work made on the project, including our academic research in various related fields, the methodologies we applied throughout the dissertation and our development process. Finally we will present our results and critically analyse them before the conclusion.

II. Research and background

We decided to review four different fields related to our subject in order to have a complete understanding of what has been done and what is important. We started with an overview of the Formal Methods and more particularly the Behavioral Interface Specification Languages since Spec# is considered as a member of this category [2]. We then analysed the presence of Formal Methods in the education, how they were taught and if there was any potential improvement suggestions for teaching materials like our library. Since our library involve data structures and algorithms, we made a quick overview of the most relevant literature in the field in order to have strong references for our implementations. Finally, since Dafny has been chosen as our specification language, we decided to review the existing literature about the language in order to have a complete understanding of its features and drawbacks.

A. Formal Methods and Behavioral Interface Specification Languages

Formal Methods have been research and used for many years in various projects, J. Woodcock, P. Larsen, J. Bicarregui et al. defines them as a set of methods using mathematical models to improve and ensure the quality of a given software engineering project through analysis and verification [3]. They made a survey of their usages in the software engineering industry and revealed a rather positive outcome regarding the project quality on various projects using different methods. As explained previously, our project will focus on a particular sub category, the Behavioral Interface Specification Languages. This category, introduced in [2] by J. Hatcliff, G. Leavens, K. Leino et al. as languages considered expressive enough to specify the functional correctness of complex programs but at the same time based on simple concepts in order to

be easily understood and usable by many industrial programmers, those languages can also be used along with automated verifiers. Their goal is to describe the behavior of a given code inside a system. *JML* [4], *Spark*, *Spec#* [5] and *Dafny* [6] are considered members of this category and seem to provide a common set of validation features including pre/post-conditions (Inspired from Bertrand Meyer's design by contract [7]), loop invariants, termination conditions, frame conditions, etc.

Many papers have been written about those languages and are mostly presented as a technical overview of the each languages, introducing point by point using code samples every specification feature. Several of them were very helpful in order to help us choose a specification language, particularly [8], [9], [6] and [10] for Dafny, [4] and [11] for JML and [5] and [12] for Spec# (we voluntarily discarded Spark since it concerns Ada, which is not a commonly used language at the M.Sc. level). As explained before, those languages share a common set of features, but we noticed some differences between them, for instance JML and Spec# are built on top of other languages (respectively Java and an old version of C#) when Dafny is a completely new language built with verification in mind (which can still be compiled to .NET-compatible code and reused). However, JML and Spec# inherit their object oriented features from their parent languages but Dafny only provides a very small support for objects and generic features, this has been pointed out by K. Leino and R. Monahan in [13] because it prevented them from creating a fully generic sorting algorithm.

The field of Formal Methods is very large, this is why we had to restrict the scope of our research to Behavioral Interface Specification Languages and more particularly Spec#, JML and Dafny. We believe that those languages are suitable in order to be used for the implementation of an educational tool, our final choice has been the Dafny language and will be detailed and explained in part III-B. This first step in our research was also helpful in order to understand the basic verifications principles and to learn the best practices regarding those languages.

B. Formal Methods in Education

In order to be truly useful, our project needs to respond to a problem regarding the presence of Formal Methods in education, in a survey realized by V. Almstrum, C. Dean, D. Goelman et al. [14], we learn that the presence is actually limited and that only some university courses include modules teaching formal methods, some of those courses only mention the term semi-formal methods. A potential reason for this problem could be the difficulty to define a context for the subject, G. Tremblay wrote that Formal Methods could be taught in the context of Mathematics, Computer Science or Software Engineering, which makes it complex to create a course suitable for all the different students [15]. He explains that the teaching of Formal Methods needs to be different depending on the course category for the student to better understand the usefulness and the concepts behind the subject.

Another problem could be related to the industry, as presented by T. Mailbaum, the actual Software Engineering Industry focuses on computer skills like programming languages instead of focusing on analytical skills, which are necessary to create an abstraction of a program and be able to use formal methods [16]. A change in the industry priorities regarding recruitment and project management could help to make Formal Methods a popular and widely used verification system.

V. Almstrum et al. also provided some improvement themes that they gathered from the experience of the members of the working group. Some themes are related to the tools used along with Formal Methods, and the general idea is that the tools could benefit from an improved user experience in order to allow the student to focus on their learning instead of tool-specific features. S. Liu, K. Takahashi, T. Hayashi et al. share the same opinion regarding the tools and explain that simple, precise and visual tools would be very helpful for the student to be interested and motivated by Formal Methods [17].

Thanks to this review, we now understand that Formal Methods need usable and helpful tools and resources for the students to go further in the field, it is also interesting to see that the industry could have an important role to play in their popularity by including Formal Methods in their processes. We used those guidelines and improvements themes during our work on the library in order to create an helpful material.

C. Data structures and their usages

Since the library will be composed of various data structures and algorithms, we need to understand their concepts and have access to a set of widely used resources in order to implement efficient and valid algorithms. Data structures are defined by C. Leiserson, R. Rivest, T. Cormen et al. as a system to store data in a way that optimizes their treatment by a computer in [18]. Their book is considered as a standard in the field and provide mathematical demonstrations, code samples, descriptions and exercises for a large set of data structures (stacks, queues, trees, sets, etc.) and other related fields. A similar, but less mathematical and more practical approach has been made by R. Sedgewick and K. Wayne in [19] by providing schemes and more code samples. Both books introduce the concept of invariants, which describe a rule that should always be valid in order for the data structure to be maintained in a correct state. This concept can be compared to the concept of Design by Contract, introduced by B. Meyer in [7], it relates to the concept of pre and post-conditions on a program method, both concepts act as rules specifying what should and should not change during an operation (Design by contract was mentioned by R. Sedgewick et al. when talking about assertions in Java).

Also, we used the documentations of existing programming platforms (Java [20] and .NET [21]) in order to design our implementations of the data structures, we wanted to provide useful methods and not only basic ones for the students to be able to relate to an existing programming language. The work of B. Long in [22] was also useful on this matter and some similarities between the proposed methods were found.

Data structures are an essential part of software engineering, we were able to understand that through the concept of Design by Contract, there is a possible relation between them and the Formal Methods, they will therefore make a perfect example for our library but it is important to respect the

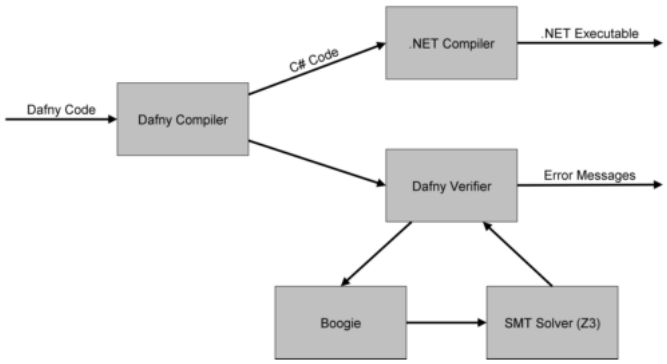

Fig. 1. The Dafny system

invariants and to provide useful methods when implementing the library.

D. The Dafny language

Finally, we reviewed the literature regarding the Dafny language, which is the chosen language for the library as explained in III-B. Dafny is presented by K. Leino in [6] as an imperative, sequential language supporting basic object features and dynamic allocations. An overview of the Dafny system has been described by L. Herbert, K. Leino and J. Quaresma in [8] and is available in figure 1, we can see that Dafny compiles the code to a .NET compliant executable but also verifies it using Boogie2, and intermediate language and Z3, an SMT (Satisfiability modulo theories), which will try to validate the program using a fixed set of decision procedures.

Many different papers are introducing the syntax and verification features of Dafny, from the pre and post-conditions to the built in data structures, they describe every feature and gives example on how to use them. K. Leino, L. Herbert, and J. Quaresma give a large overview in [6] and [8] and summarize a large part of the language along with many code samples. The framing system in Dafny is called Dynamic Frames and is here to specify which parts of the object can me modified and which parts can't, an introduction to dynamic frames has been given by I. Kassios in [23] and the implementation in Dafny has been discussed in [24], we can see that there are many similarities, which allowed K. Leino to explain how to use this system within Dafny.

Even if we use the object oriented part of Dafny in our project, there is a way to specify abstract and refined modules using the refinement system in Dafny. This is introduced by J. Koenig and K. Leino in [25] through a set of examples featuring a nearly generic sorting algorithms (which we weren't able to implement).

Some comparisons with similar languages like JML and Spec# have been made by K. Leino in [6] and by R. Monahan and K. Leino in [13], they concluded that Dafny is a decent verification language including a complete set of verification features but a very light object oriented system. On the performance side, Dafny was able to prove complex specifications relatively quickly as shown in [13]. Those articles will help us to provide efficient Dafny specifications in the library and to have a better understanding of the language's features and syntax.

III. Development approach

Most of the time allocated to the project was taken by the development phase, composed of a feasibility study, a design analysis, a specification/implementation phase and a testing phase. In this section we will summarize this process and outline the main points and choices made during the project. Finally, a case study of the linked-list-based Queue implementation will be presented.

A. Requirements and constraints

This project was specified using three categories of requirements, the first ones are the functional requirements. The library is expected to contains at least the Stack, Queue, TreeMap and HashMap data structures, its source code must be clean, extensible and commented and can be entirely verified using the automated verifier of the specification language. We also had some non-functional requirements specifying that the library must be developed using a specification language which should provide a syntax close to a common programming language like Java, C# or even C. The library's correctness must be verifiable using and external tool and the algorithms and data structures complexity must match the current industry standards. Finally, some extended requirements were proposed, mentioning that the library should be compilable and/or reusable, could include more algorithms/data structures if relevant and could provide guides for the library usage and the environment configuration.

We detected two main constraints at the beginning of the project, the first one being the allocated time-frame, the project started around May 2016 and had to be finished by the 30th of September 2016, it was enough time but it was important to take it in account in order not to waste time on non-relevant parts. The second constraint was a technical one and was related to the chosen specification language, in our case, Dafny. We knew at the beginning that Dafny has a weak object oriented support, but we weren't sure if it was going to be a problem for the design of the data structures. It actually prevented us from creating fully generic version of some algorithms, a mitigation plan was applied to limit the impact of this obstacle.

Aside of those constraints, some risks were expected during the project, the first one concerning the Dafny language and its limitations, which, as explained before, caused an issue. The mitigation solution for this problem was to accept the implementation of a non generic version since we believed it was enough for the educational purpose of the library. The second risk was related to our specifications and implementation skills and the possibility that we fail to fully validate some parts of the library, this risk also caused an issue for the HashMap, this is the only data structure that we failed to fully validate, our mitigation plan for this issue was to accept the problem and to add the List data strucure to the library to provide enough materials for the students. Three other minor risks were planned, regarding the correctness of the implemented data structures, but we didn't detected any problems in our implementations, another one concerned the ethical problem

of making a personal choice instead of a well justified choice, which could lead to a wrong influence on the students for a particular technique, we believe that we also avoided to fall into this one since our choices are all justified by precise comparisons and academic references. The last risk can't be fully evaluated at this time and concerns the usefulness of our work in an educational situation, it would be very interesting to see if the project fits correctly in a Formal Methods module, but this can't be correctly assessed at the time of writing.

B. A critical choice: the specification language

The main choice that we made in the project was the choice of the specification language. As explained previously, during our literature review, we decided to choose between JML, Spec# and Dafny in order to stay close to the previously used language in the Oxford Brookes University module, Spec#. All those languages are offering and automated verifier and decent verification features, but they have a few differences. We will present a summary of our analysis to explain our choice of the Dafny language.

Spec#, the current language, is built as a layer of an old version of the C# language, it provides nearly the same syntax, and the essential specification features. A web-based IDE is even provided in order to try it quickly, it is also possible to install it on a computer but it requires an old version of Visual Studio. The main problem related to Spec# is that the project is not active anymore as explained by the community coordinator, K. Rustan M. Leino in a forum thread [26], he advices instead to use Code Contracts with .NET or the Dafny language.

JML is built as a layer of the Java language and therefore shares its syntax with a few added elements for the verification purposes, it also provides the essential verifications features and can be compiled. Regarding the usability, we didn't managed to get a version of JML working on multiple configurations and since it doesn't provide a web-based IDE it was difficult to test it completely.

Dafny is the only language to be built from scratch as a verification language, he shares some syntax elements with classical languages like C# and Java, but also with some functional languages for the verification and mathematical expressions. A web based IDE is available and a desktop version can also be installed even if it requires Visual Studio 2012 (the documentation mentions a possibility to install it on other systems than Windows but we didn't tried to).

We decided to choose Dafny because 1: it is easily installable and/or usable on the web based IDE, 2: the syntax is close to the previously used language, 3: the project is still active and could receive more improvements. Those reasons make us believe that Dafny will be a useful language for an educational tool and that it will help students to provide better and more complex specifications.

C. Methodology and development process

Regarding the project management, we used a simple task-based methodology. We maintained a kanban board with three columns, the tasks to do, the active tasks and the finished tasks, the to do column often had a month load of tasks ahead of the current date in order to have a correct estimation of the project advancement. One or two tasks were usually active at the same time, most of the time a task of research or a task of writing along with a task of implementation in order to keep a smooth progress in the project. We had three main different phases during the project, the specification and research phase which took approximately a month and a half, we used this time to make the initial literature review, draw the first requirements and write the proposal report. The second phase was the implementation one, during this time we finished our research with additional papers and we designed, specified and implemented step by step the whole library. This phase took two months and required intensive work. The final phase was started at the end of August and was the testing and reporting phase, we took enough time to assess and completely test the project, then to assemble all of our notes in order to write the final reports.

Regarding the development process, an initial design phase was made before starting the data structures specifications in order to create a structure for the library and to define the first data structures interfaces and methods. When this phase was done, we followed the same process step by step for every one of the data structures and algorithms. We first made a small design review in order to be sure that the interface was correctly designed and that it would not cause any problem during the implementation, then we started the specification. The specification phase was only made to write the description of what the system was expected to do using Dafny without even writing any implementation part. When this was done, we started the implementation by filling the content of the methods with the actual implementation code. Most of the time the implementation caused problems and did not fit exactly with the specification, when the issue happened, we used a combination of assumptions and assertions to find the problematic part and we modified either the specification or the implementation (depending on where the error was located) for the program to be fully validated. We will provide a detailed example with the work made on the linked list-based queue in this paper.

D. Testing and quality assessment

Since this project involves the use of Formal Methods, the testing part is a little bit different from classical software engineering projects, we are using a combination of manually written test cases and automated verification in order to assess the correctness of our implementations. When a part of the library is implemented, our first verification is to ensure that the automated verifier validates the implementation, if it is the case it means that our implementation is correct towards our specification. However, we can't be sure that our specification will match our requirements as users of the data structure, that is the reason why we also wrote a manual test case using all the available methods in a different order along with assertions to verify that the return values are correct and match our expectations. This two-way testing procedure has helped us to detect many errors in our specification even if Dafny recognised our implementation as valid. Also, we followed our own coding standards throughout the library which are based on the existing code samples that we found during our literature review, our comments are placed in a way that it

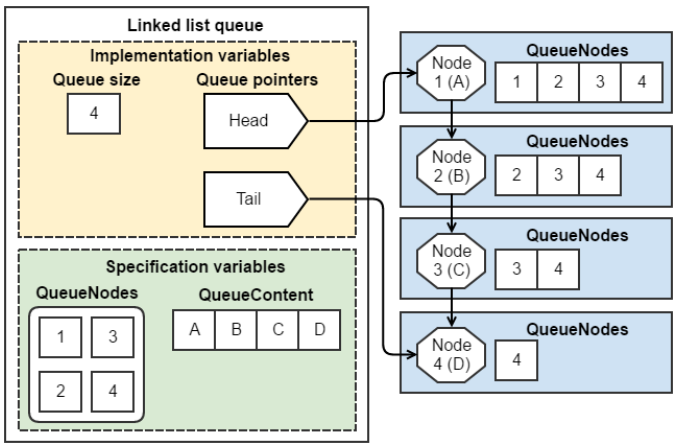

Fig. 2. The queue specification and implementation variables

doesn't affect the readability of the program and we made sure that only comments providing useful help were kept.

E. An example: the linked list-based queue

In order to present a real example of our implementation process, we will detail the implementation of the linked list-based queue in our library. The initial design phase covered the main parts of the Queue, we decided to provide the following methods : *enqueue*, *dequeue*, *isEmpty*, *size* and *peek*.

We then moved on to the specification phase, we wanted to be able to not only prove that the method are valid, but also that at any time, the state of the Queue object is valid. That is the reason why, as seen in the figure 2, we are using an exact mirror of the actual linked list as specification variable to verify that at all times the actual linked list matches the specification variable *QueueContent*. This specification variable is a sequence, which is a data structure built-in the Dafny language. We are also using a set, *QueueNodes*, which is also a verification variable, in order to keep track of all the nodes in the list. The verification of the queue proceed by checking that every node in the set is valid towards its next nodes, since this verification is made for every one of the nodes, we can be sure that the queue is valid. In order to access all the nodes of the linked list at any time, we are using a dynamic frame (not shown in figure 2) which keep tracks of all the objects involved in the queue in order to simplify the code. Regarding the implementation, the queue was relatively straightforward to implement since we just have to manage a linked list, the complex point is to maintain the verification variables in the same state than the linked list. In order to ensure this we have to manually update the verification set and sequence for every *enqueue* and *dequeue* operation.

We had many problems to completely prove the queue before validating the TreeMap, we used at first a recursive validation of the linked list, but we didn't managed to prove it this way because we were not able to completely prove the validity of our recursive validation method. The work realized on the TreeMap with the dynamic frames helped a lot to fix the linked list, we decided to try a non recursive version along with the modifications that we made on the dynamic frame in the TreeMap to validate the queue and it was much more easier to prove valid.

TABLE I. OVERALL COMPLETION OF THE LIBRARY

Data structure / Algorithm	Validated	Tested	Comments
SimpleStack	Yes	Yes	Generic
ArrayStack	Yes	Yes	Generic
LinkedListStack	Yes	Yes	Generic
LinkedListQueue	Yes	Yes	Generic
ArrayList	Yes	Yes	Generic
KeyValue LinkedList	Yes	Yes	Generic for values only, not keys
TreeMap	Yes	Yes	Generic for values only, not keys
HashMap	**Partly***	**Yes***	Generic for values only, not keys; * Validated and tested using assumptions
BubbleSort	Yes	Yes	Not generic
InsertionSort	Yes	Yes	Not generic

The test case is not shown in this report, but it is not particularly interesting since it is only composed of calls to the methods listed earlier along with assertions used to verify that the queue behaves correctly.

IV. RESULTS

We will now present our result and how did we measured them, a global overview of the project completion will be given followed with a summary of our contribution to the field of Formal Methods.

A. Overview of the completion

Regarding the library, nearly everything has been successfully completed as shown in table I. We can see that all the algorithms and data structures are validated (using the Dafny automated verifier) and tested (using the manually written test cases) except for the HashMap which is, as explained earlier, only partially validated thanks to Dafny assumptions. Some implementations fully support generic types except the sorting algorithms which are limited to an integer type in our implementation because of the limits imposed by the Dafny language regarding object oriented design. The key-based data structures (HashMap, TreeMap and KeyValueLinkedList) only have a partial support for generics and can only handle generic values, not generic keys because of the same problem, we can't compare generic keys using Dafny.

The project however shows its limits at the time of writing since it is difficult to make it evolve towards an industrial version which would require a full generic support of the data structures. An improvement of the Dafny language regarding the object oriented support would allow to propose an evolution of the library featuring a fully generic version. More work is also required on the HashMap in order to provide a fully validated version.

We accepted the issues that we mentioned in this paper and proposed mitigation solutions (like the insertion of the List data structure) to make up for the lack of content caused by those issues. We believe that even with the issues encountered during the project, the majority of the project has been successfully completed and that the library will be a useful tool for students in Formal Methods modules.

B. Contribution to the field

We think that our project will contribute to the field of Formal Methods on several levels. First we believe that it will actually help students to have a better understanding and an increased interest towards the field by having access to the concrete examples provided in this library. Increasing the interest in Formal Methods could contribute to an increased usage and therefore an improvement for the formal methods usage in the education or the industry. Also, our literature review summarizes a set of writings about Formal Methods and more particularly Behavioral Interface Specification Languages and their usage in education, which can be helpful for someone looking for an overview of the field. Finally, our project features a nearly fully proven data structure library which is also compilable and reusable in .NET programs, as mentioned earlier, we did not find any paper mentioning discussing a similar project, it can therefore be an example that widely used libraries could use Formal Methods in order to increase their reliability level.

V. Conclusion

As a conclusion we propose a critical evaluation of our work during the project in order to highlight the key successes and failures of our methodology. We will also discuss the potential applications of our project and its limitations in different contexts before opening with a suggestion of future work directions that could continue to improve the field of Formal Methods.

A. Critical discussion

Globally speaking, we believe that the goal goal of our project, creating a formally proven data structure library intended for Formal Methods teaching, has been successfully reached. The library provides a large set of documented data structures featuring different implementation and specification techniques using the Dafny language for the students to use in order to help them with the Formal Methods. Our work also features a complete and detailed literature review of the fields related to our project, which was very helpful for us during our work and could be useful to other people wishing to work in the field of Formal Methods in education. Our project management was very effective, the concept of the kanban board was really helpful in order to keep a constant progress in our work and to visualise the progress and the remaining work load. However, some points could have been improved, for instance our management of the time allocated to the project. We decided to change our initial planning, which included a time for redaction after each implementation of a data structure / algorithm to a single writing phase at the end of the project. We made that choice because we thought that it would be easier to concentrate on the implementations and then to take a decent period of time to write the report. We now understand that it would have in fact been easier to write directly after the implementation in order to remember everything easily and not have to remember the whole project exactly at the same time. We think that it would also have left more time to finish the validation of the HashMap.

Regarding the implementation, we think that studying SMT solvers (like the Z3 program used in Dafny) would have helped in order to fully validate the HashMap by having a better understanding of how Dafny validates the specification, unfortunately we only thought about that point at the end of the project. Minor issues due to choice of the language have been mentioned and even though they limit the scope of the library, we don't see them as a failure, we think that the parts affected by those issues are still valuable and will contribute to the library's goal.

B. Potential applications

The main expected usage of the library is education and we believe that it will be very helpful for the students to understand and have access to large examples and for the lecturers to use the material in their modules. Aside of this main purpose, an industrial use could be possible, during our research, we were not able to find an existing data structure library intended for industrial projects which features formal specifications. Our project could be useful to increase the reliability of such libraries, it would however require a better generic support for the data structures which could be achieved if Dafny were to support more generic object oriented features.

C. Future work

Even if working on this project was a very interesting experience, we did notice some particular points in the field of Formal Methods which could be improved. The first one being the traits system in Dafny, which offers a large possibilities of object oriented design. This system, introduced by R. Ahmadi, K. Leino and J. Nummenmaa in [10] doesn't support generic types, which has been a problem for us with the library, contributing to the Dafny language in order to add support for this issue could make Dafny allow the creation of even more object oriented systems with Dafny and opens it to industrial usages. More generally speaking, we saw in the literature review that the field of formal methods could benefit from improved tools, we would have liked to have access to a dedicated development environment for Dafny or similar languages during our project. The environment could provide many useful features such as syntax colouration, code completion, embedded compilers, verifiers and debuggers, etc. The easiness of use and efficiency of the tools would have to be the main goals in order for the project to be really useful in educational and industrial usages.

Acknowledgment

I would like to thank my supervisors, Ian Bayley, for the support and the helpful answers he gave me throughout my dissertation and for the very interesting lectures and advices I received from him and David Lightfoot on Formal Methods which motivated me to choose this subject for my dissertation. I am also grateful to Chris Cox, for his lectures on academic writing and other subjects, and his kind advices regarding my writings in this dissertation.
Finally, I would like to thank my father, Jean-Marc Chevalier, for this advices on my dissertation and my whole family for their unconditional love and support throughout my life and studies.

REFERENCES

[1] "Oxford Brookes University - Department of Computing and Communication Technologies Postgraduate Modules," https://students.cct.brookes.ac.uk/PG_Modules, [Online; accessed 21-June-2016]. [Online]. Available: https://students.cct.brookes.ac.uk/PG_Modules

[2] J. Hatcliff, G. T. Leavens, K. R. M. Leino, P. Müller, and M. Parkinson, "Behavioral interface specification languages," *ACM Computing Surveys*, vol. 44, no. 3, pp. 1–58, 2012.

[3] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods," *ACM Computing Surveys*, vol. 41, no. 4, pp. 1–36, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1592434.1592436

[4] G. T. Leavens, A. L. Baker, and C. Ruby, "Preliminary design of JML," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 3, p. 1, may 2006. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1127878.1127884

[5] M. Barnett, K. R. M. Leino, and W. Schulte, "The Spec\# Programming System: An Overview," *International Conference in Construction and Analysis of Safe, Secure and Interoperable Smart Devices (CASSIS '04)*, no. October, pp. 49–69, 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.2133

[6] K. R. M. Leino, "Dafny: An automatic program verifier for functional correctness," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6355 LNAI, pp. 348–370, 2010.

[7] B. Meyer, "Applying "Design by Contract"," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[8] L. Herbert, K. R. M. Leino, and J. Quaresma, "Using Dafny, an Automatic Program Verifier," pp. 156–181, 2012. [Online]. Available: http://link.springer.com/10.1007/978-3-642-35746-6_6

[9] K. R. M. Leino, "Developing verified programs with Dafny," *Proceedings - International Conference on Software Engineering*, pp. 1488–1490, 2013.

[10] R. Ahmadi, K. R. M. Leino, and J. Nummenmaa, "Automatic verification of Dafny programs with traits," in *Proceedings of the 17th Workshop on Formal Techniques for Java-like Programs - FTfJP '15*. New York, New York, USA: ACM Press, 2015, pp. 1–5. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2786536.2786542

[11] G. Leavens and Y. Cheon, "Design by Contract with JML," *Draft, available from jmlspecs. org*, vol. 1, p. 4, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.7152

[12] M. Barnett, M. Fähndrich, K. R. M. Leino, P. Müller, W. Schulte, and H. Venter, "Specification and verification: The Spec# Experience," *Communications of the ACM*, vol. 54, p. 81, 2011. [Online]. Available: http://portal.acm.org/citation.cfm?id=1953145

[13] K. R. M. Leino and R. Monahan, "Dafny Meets the Verification Benchmarks Challenge," 2010, vol. 6217, no. June, pp. 112–126. [Online]. Available: http://link.springer.com/10.1007/978-3-642-15057-9 http://link.springer.com/10.1007/978-3-642-15057-9_8

[14] V. L. Almstrum, C. N. Dean, D. Goelman, T. B. Hilburn, and J. Smith, "Support for teaching formal methods," *ACM SIGCSE Bulletin*, vol. 33, no. 2, p. 71, jun 2001. [Online]. Available: http://portal.acm.org/citation.cfm?doid=571922.571962

[15] G. Tremblay, "Formal methods: mathematics, computer science or software engineering?" *Education, IEEE Transactions on*, vol. 43, no. 4, pp. 377–382, 2000. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=883345

[16] T. Maibaum, "Formal methods versus engineering," *ACM SIGCSE Bulletin*, vol. 41, no. 2, p. 6, jun 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1595453.1595455

[17] S. Liu, K. Takahashi, T. Hayashi, and T. Nakayama, "Teaching formal methods in the context of software engineering," *ACM SIGCSE Bulletin*, vol. 41, no. 2, p. 17, jun 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1595453.1595457

[18] C. C. E. Leiserson, R. R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to Algorithms, Third Edition*. The MIT Press, 2009. [Online]. Available: http://www.amazon.com/Introduction-Algorithms-Third-Thomas-Cormen/dp/0262033844

[19] R. Sedgewick and K. Wayne, "Algorithms (4th Edition)," in *Algorithms (4th Edition)*, 4th ed. Addison-Wesley Professional, 2011, pp. 1–992.

[20] "Collections in Java - Oracle Java Documentation," http://docs.oracle.com/javase/tutorial/collections/, [Online; accessed 22-June-2016]. [Online]. Available: http://docs.oracle.com/javase/tutorial/collections/

[21] "Collections and Data Structures in .NET - Microsoft Developer Network," https://msdn.microsoft.com/en-us/library/7y3x785f(v=vs.110).aspx, [Online; accessed 22-June-2016]. [Online]. Available: https://msdn.microsoft.com/en-us/library/7y3x785f(v=vs.110).aspx

[22] B. Long, "Towards the design of a set-based Java collections framework," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 5, p. 1, oct 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1838687.1838705

[23] I. T. Kassios, "Dynamic Frames and Automated Verification," pp. 1–26, 2011.

[24] K. R. M. Leino, "Specification and verification of object-oriented software," *Engineering Methods and Tools for Software Safety and Security, lecture notes of the Marktoberdorf International Summer School 2008*, vol. 22, pp. 231–266, 2009. [Online]. Available: http://books.google.com/books?hl=en&lr=&id=L4iMiT84pOsC

[25] J. Koenig and K. R. M. Leino, "Programming Language Features for Refinement," *Electronic Proceedings in Theoretical Computer Science*, vol. 209, pp. 87–106, jun 2016. [Online]. Available: http://research.microsoft.com/en-us/um/people/leino/papers/krml248.pdf http://arxiv.org/abs/1606.02022

[26] "Spec# vs dafny," https://specsharp.codeplex.com/discussions/569610, (Accessed on 09/21/2016).