Can Formal Methods Make Automotive Business Sense?

A CLASSIFICATION OF FORMAL METHODS BY USEFULNESS

Pat McElligott, Anila Mjeda, Steffen Thiel
Lero – The Irish Software Engineering Research Centre
University of Limerick, Ireland

ABSTRACT

Legislative bodies are directing that automotive products comply with stringent safety levels. The liability for the safety of passengers in an automobile has traditionally been quite complex. Other transport sectors are externally regulated, and liability lies with the manufacturer or the transport service provider. The automotive industry is self-regulated and the individual driver carries a significant liability.

Software and electronics increasingly provide greater control of automotive safety, possibly reducing driver liability, and increasing the need for more formal software development methods. The automotive business model, however, also presents challenges to the effective use of formal methods. An automotive design change costing €600 per vehicle could consume 100% of gross margin. In aviation, this cost represents 0.01% of gross margin. [1] [2].

The automotive industry is responding to the increasing impact of automotive software with the development of standards such as AUTOSAR [3], and EU funded projects such as ATESST [4] and EASIS [5]. They propose architectures which might deliver the benefits of best software engineering practice to the industry. In terms of safety, they recommend existing accepted standards such as IEC61508 [6], which stipulates various formal methods for the development of safety-critical software. However, IEC61508 does not compare specific formal methods in terms of their suitability to industry.

This paper discusses the suitability for industry of formal methods of specification and verification. It provides a classification which looks at categories such as commercialization; capacity to solve industry-scale problems; cost effectiveness, etc. The paper looks at the relevance of the classification in terms of the challenges and constraints of the automotive domain and discusses how it might facilitate the engineer to make design decisions which improve safety in a cost effective manner.

INTRODUCTION

The customer and the legislator are the two major stakeholders in the determination of automotive requirements. The market-place tends to be interested in comfort, general reliability, safety and fuel efficiency. Legislative bodies at national, regional and global levels are directing that automotive products comply with stringent safety and reliability levels as well as attaining high fuel efficiency and low emission levels [7] [8].

Developing reliable, safe automotive systems, however, is difficult because of the inherent heterogeneity of ECUs and communication networks. The chassis systems, for example, handle an extensive amount of inter-process communication between a large number of internal software functions as well as communicating with the powertrain and body systems through numerous interfaces. The sensors and actuators of the chassis are widely distributed spatially in the vehicle and as a result there tends to be a higher number of ECUs in the configuration of systems, such as, ABS, Throttle control (TCS), ESP [9]. Networks and operating systems of the automotive vehicle add further levels of complexity to the architecture.

Industries such as aviation and rail grapple with similar challenges involving the safety of complex heterogenous distributed embedded systems. They have responded with heavy investment in the use of software engineering methods including formal methods.

This paper argues that one must look at the potential use of formal methods in the automotive industry primarily in the light of automotive business criteria. To do this, one needs to be able to classify formal methods
in terms of criteria that make sense to the automotive software engineer. The software engineer is a key stakeholder and supplier of decision-making information in any adoption of formal methods. Although Annex A of Part 3 of IEC61508 [6] says that formal methods for specification of software safety requirements at SIL4 are “highly recommended” and they are still “recommended” at SIL3, it does not compare specific formal methods in terms of their suitability to industry.

This paper is divided into the following sub-sections. A brief description is first given of automotive business criteria that should be considered, when selecting formal methods and tools for industry-scale use. This is followed by a brief explanation of our research method. Then a description and evaluation is given of existing related research into the classification of formal methods in terms of usefulness. Finally, a proposed classification of formal methods by usefulness is defined, based on a large sample of specific methods and tools, which is then quantified to facilitate comparison, evaluation and recommendation.

AUTOMOTIVE BUSINESS CRITERIA AND FORMAL METHODS

Cost, liability, and system complexity contribute significantly to the difficulty of achieving and improving safety levels, in the automotive industry. In the following we describe the nature of these difficulties, and how work measurement, standardization and scalability of formal methods alleviate them.

COMPLEXITY AND SCALABILITY

System complexity is illustrated by the multiplicity of networks needed to meet the distributed functionality required in vehicles today. Networking requirements are varied and complex in response to the competing demands of cost, vehicle space, safety and reliability and robustness in a harsh environment. Some functions of the body systems such as central locking, for example, require the coordination of variables and actions to do with multiple ECUs in the group, and this is greatly facilitated by robust and reliable network systems such as CAN, LIN [9]. On the other hand, a specialised high-speed network is needed to cater for large voice and video data requirements of multimedia options such as, tuner, audio system, video system, navigation system, telephone. Network protocols such as MOST offer speeds ranging from 25Mbit/s to 50 Mbits/s over a combination of optic fibre and twisted pair physical media, offering synchronous guaranteed bandwidth to ensure jitter-free quality of service [10]. To use formal methods with complex systems, the software engineer must first assess the scalability of specific tools and methods.

LIABILITY AND STANDARDS

The liability for accidents may not be as clear-cut in automotive industry as in similar industries, such as aviation and rail where, liability lies much more clearly with the manufacturer or the transport service provider. The liability for the safety of passengers in an automobile has traditionally been quite complex. The automotive industry is self-regulated and the individual driver carries a significant liability. The balance of liability may change as software and electronics increasingly provide greater control of automotive reliability and safety. This may also effect the automotive industry’s perspective on the use of formal methods. A software engineer may need to assess how well a particular formal method matches up to certification and standards compliance requirements.

COST MODELS AND FORMAL METHODS METRICS

The automotive business model with “cost per piece and production-centric cost models” [11] has not generally found the cost/benefit justification to adopt formal methods for software development. The automotive industry business model is characterised by low volumes and high margins as illustrated in Figure 1. The two highest ranking vehicle manufacturers in the world and the top European manufacturer combined, produced a total of over 21 million vehicles in 2005. Their gross margin, however, only ranged from 3-5% or €350-892 per vehicle [2, 12, 13, 14].

Their research and development teams, as a result, train a much more critical eye on product cost and opportunities to reduce cost throughout the product lifecycle, than similar industries such as rail, shipping and aviation. Boeing, for example, and the manufacturer of Airbus, EADS, have a gross margin per product unit of over €6 million [1, 15, 16]. An automotive design change costing €600 per vehicle could consume 100% of gross margin. In aviation, this cost represents 0.01% of gross margin. Attending to the control of other costs such as software development, maintenance and the failure to reuse software components may furnish the automotive industry with the cost/benefit justification to employ software engineering best practices [17] including formal methods. A prerequisite of this is that any candidate tool or method must provide accurate and detailed measurement of formal methods work units so that the benefit of adopting them can be quantifiably expressed.

<table>
<thead>
<tr>
<th>Prod Cost</th>
<th>Sales</th>
<th>Unit Sales</th>
<th>Gross Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>€ billion</td>
<td>€ billion</td>
<td>€ billion</td>
<td>% of sales</td>
</tr>
<tr>
<td>GM</td>
<td>121</td>
<td>124</td>
<td>9 million</td>
</tr>
<tr>
<td>Toyota</td>
<td>121</td>
<td>128</td>
<td>7 million</td>
</tr>
<tr>
<td>VAG</td>
<td>82</td>
<td>92</td>
<td>5 million</td>
</tr>
</tbody>
</table>
Specific methods and tools were assessed based on their websites, most of which contained good explanatory material and introductory papers as well as indication of types and results of usage. Information was extracted about the taxonomy of the methods, industrial usage, and future directions. The information was supplemented where appropriate with assumptions and inferences which may not be attributable to the actual source, but which would be regarded as generally reasonable and interesting to propose.

A quality checklist, detailed in the main section of this paper, was applied to specific formal methods based on Bowen and Hinchey’s “10 Commandments” of successful application of formal methods [19] [20].

RESEARCH METHOD

This paper attempts a classification of formal methods in terms of industrial usefulness by researching the following question: Which formal methods maybe suitable in the domain of automotive embedded distributed software development? The research method adopted is based on guidelines for conducting research reviews, described in [18]. Although a full ontological classification of methods is beyond the resources of the present effort, it is hoped at least that it provides an extra level of detail beyond the general classification of formal methods into specification and verification methods. Evidence is gleaned from published papers, web databases, etc rather than from interviews, questionnaires etc. Criteria to be considered and which should help scope and further define the question are as follows:

- A key criterion for usefulness is how effectively a method works in industry. Would software engineers be likely to adopt the method? Would introduction of the method to a work environment be likely to have a negative or positive impact overall?
- Another important criterion is whether a formal method addresses special needs of an industry. For example, if a formal method does not cater for concurrency then it may be less useful, or constrained in its usage.
- A formal method may theoretically solve certain problems of correctness but may not scale to an industrial problem.
- The work effort involved in a method may cancel financial or project timeliness benefits.
- Industry take-up of a method may be forestalled by lack of tools and commercialisation

We employed a modus operandi, which aimed to confine the search to top-tier journals (IEEE, ACM, Elsevier and LNCS) and recent books and chapters no earlier than year 2000 in most cases, which survey formal methods, with particular attention to surveys by authors with a long standing reputation in the field.

EXISTING CLASSIFICATIONS OF FORMAL METHODS

FMEA

The formal Methods Europe Association (FMEA - http://www.fmeurope.org/) classifies formal methods in [21] according to what courses are taught in Europe. They followed a bottom-up approach, examining the courses taught and categorising them into broad sub-types of formal methods as shown in Figure 2 above. They attempt to establish what is perceived to be the necessary ‘book of knowledge’ constituting an education in formal methods, as reflected in university courses currently offered throughout Europe. The sub-typing reflects their preoccupation with what a student should know, rather than what methods are available to the future engineer. Formal semantics, for example, is a frequently taught topic, as are tools that support executable specification. From the viewpoint of usefulness however, one would expect semantics to be subsumed under usage sub-types as a property. It might be justifiable to use specification executability as a sub-type of usefulness, but foundational topics are clearly of relevance only to classification of what types of formal method subject-matter should be taught.

A key criterion for usefulness is how effectively a method works in industry. Would software engineers be likely to adopt the method? Would introduction of the method to a work environment be likely to have a negative or positive impact overall?

Another important criterion is whether a formal method addresses special needs of an industry. For example, if a formal method does not cater for concurrency then it may be less useful, or constrained in its usage.

A formal method may theoretically solve certain problems of correctness but may not scale to an industrial problem.

The work effort involved in a method may cancel financial or project timeliness benefits.

Industry take-up of a method may be forestalled by lack of tools and commercialisation

We employed a modus operandi, which aimed to confine the search to top-tier journals (IEEE, ACM, Elsevier and LNCS) and recent books and chapters no earlier than year 2000 in most cases, which survey formal methods, with particular attention to surveys by authors with a long standing reputation in the field.

RELATED WORK

FMEA

The formal Methods Europe Association (FMEA - http://www.fmeurope.org/) classifies formal methods in [21] according to what courses are taught in Europe. They followed a bottom-up approach, examining the courses taught and categorising them into broad sub-types of formal methods as shown in Figure 2 above. They attempt to establish what is perceived to be the necessary ‘book of knowledge’ constituting an education in formal methods, as reflected in university courses currently offered throughout Europe. The sub-typing reflects their preoccupation with what a student should know, rather than what methods are available to the future engineer. Formal semantics, for example, is a frequently taught topic, as are tools that support executable specification. From the viewpoint of usefulness however, one would expect semantics to be subsumed under usage sub-types as a property. It might be justifiable to use specification executability as a sub-type of usefulness, but foundational topics are clearly of relevance only to classification of what types of formal method subject-matter should be taught.

Figure 2. FMEA Classification of Formal Methods

SWEBOK

The Software Engineering Book of Knowledge (SWEBOK) [22] in Figure 3, attempts a more classical classification. The sub-types of specification languages may not be disjunctive: it is not clear how ‘behavior-
oriented' specification is distinct from both a model- and property-oriented specification. Wing in [23] describes how both model- and property-oriented sub-types are used to describe many different types of system behaviour. Many definitive text books adopt this subdivision of specification into two broad sub-types: model- and property-oriented [23, 24, 25, 26] As a general classification it is more applicable as a means of communicating the broad types of formal method that exist and could be referenced for descriptive purposes, in a more detailed classification of specific methods by usefulness. The SWEBOK classification does not provide any reference to specific methods nor does it attempt to break the classification down into lower subdivisions such as the techniques and measures listed in Annex B Part 3 of IEC 61508 [6].

Figure 3. SWEBOK Classification of Formal Methods

Behavioral Classification

Although behavioural classification is not, to my knowledge, presented formally in the literature, it is quite common to find it informally used. Sub-types such as sequential, concurrent, temporal, hybrid, safety-critical, performance, structural, resource, are frequently used to typify formal methods and tools [27]. However, this classification doesn’t necessarily enlighten the software engineer about the quality of the method: it simply informs her that the method claims to formalise the definition of this kind of system behaviour.

Evaluation

FMEA classification is oriented towards the teaching of formal methods. The categories Foundations and Formal Semantics, for example, would not help a software engineer in industry to differentiate methods and tools by usefulness. The SWEBOK classification is at a very high-level and it is not disjunctive from the viewpoint of usefulness: a particular method is likely to appear under more than one sub-type. Finally, the classification by behaviour is, by definition, descriptive of intended use, but this classification alone does not help the software engineer to select a method which is scaleable, commercialised, cost-effective, etc.

PROPOSED CLASSIFICATION OF FORMAL METHODS BY USEFULNESS

INTRODUCTION

As illustrated in the previous section, existing classifications of formal methods do not adequately help the software engineer to select useful formal methods and tools for the purposes of enhancing company or project software engineering practices. It is not possible to quickly sift through hundreds of candidate methods. Websites, introductory and tutorial papers may be couched in language that the software engineer cannot quickly evaluate. The proposed classification attempts to alleviate this problem by matching formal methods to criteria that are commonly understood and applied in industry. Generally, software tools are selected on the basis of criteria such as scalability, cost effectiveness, ease of use, etc. Much the same kind of criteria can be applied to the selection of formal methods, as argued convincingly by Bowen and Hinchey during the last decade [19, 20, 28, 29]. This paper uses the criteria listed below, most of which partly correspond to Bowen and Hinchey’s “10 Commandments” of successful application of formal methods. The numbers in brackets after each checklist item correspond to Bowen and Hinchey’s “Commandments”.

- Has the method been successfully commercialized?
- Does the method assimilate well with the development cycle and other formal methods? (5)
- Does the method scale to industry problems, or it versatile enough to deal with concurrency, real-time, hybrid ….? (1)
- Does the method demonstrate cost effectiveness? Is there less testing but more focused testing or less field failure? (3,9)
- Can significant benefit be gained up front and/or by reuse and/or formal requirements analysis? (2,6,8,10)
- Is the method a recognised standard or has it been used for certification? (7)

Missing from this list is the fourth ‘Commandment’ that the method must be mathematically grounded. For the purposes of this paper we are assuming that the formal methods reviewed fit this requirement. We focus more on differentiating particular tools and methods on the basis of business criteria of usefulness. In 1990 Wing captured the dichotomy between rigid adherence to the mathematical basis of formal methods and the looser, more synergistic relationship between formal methods and their usage : “For a method to be formal, it must have a well-defined mathematical basis. It need not address any pragmatic considerations, but lacking such considerations would render it useless. Hence, a formal method should possess a set of guidelines or a “style sheet” that tells the user the circumstances under which
the method can and should be applied as well as how it can be applied most effectively.” [23].

The analysis of formal method and tool instances that follows, attempts to provide such a guideline showing how well formal methods performs against the listed criteria. It is not intended to be an exhaustive listing of all methods and tools. Methods and tools cited are representative and are cited to exemplify the benefit of such a classification.

COMMERCIALISATION

The degree to which formal methods tools have been marketed as products with reliable support and service is a likely prerequisite for adoption by industry. A very small percentage of the methods and tools surveyed qualify for mention here. Commercialisation of tools and methods is hampered by the lack of openness in most tools: the technologies are trapped inside the tools rather than exported to their APIs and there are no users for the tools in many cases. Rushby calls this a ‘failed market’ in verification technology [30].

One frequently quoted success is the B-method whose commercialized toolsets B-Atelier [31] and B-Toolkit [32] were developed in the early 90s. However, the B Method has not had the kind of widespread adoption that it was expected to have. Many prospective users have been put off by the need to build up tens of thousands of proofs to gain the benefits of the method. One of the longest established formal methods companies, Praxis [33], provides a language and tools for static checking and system proving, which have been employed extensively in aviation and rail systems including systems which require SIL4 safety certification. Esterel’s SCADE product [34] is based on the synchronous and data-flow oriented Lustre programming language [35] and generates C and Ada. It is used to develop the components of products for many major companies including Airbus, Schneider Electric, and Eurocopter. Both Praxis and Esterel have established themselves commercially as leaders and specialists in the field of development and certification of safety-critical systems.

Escher Technologies [36] has recently developed a commercial specification language for object oriented software with automatic prover called Perfect Developer, which has had some industrial use to date. Scepticism about the safety and reliability of object oriented approaches may mitigate against its rate of adoption. The extent of the success of this tool will not become clear for a number of years: Escher is still developing Perfect Developer’s concurrency and real-time capabilities.

An alternative transfer path to commercialization is distribution through open source communities. ESC/Java2, for example, has had some success with this approach, and cites the following companies: DoCoMo Labs, RapidMoney Corporation, Sun Microsystems, ACME Labs [37].

ASSIMILATION

A software engineer assessing formal methods tools is likely to be persuaded by the flexibility of linking formal requirements to a model checker, or probabilistic state machines to test generation. Many formal methods, however, tend to be stand-alone, without facility to connect with other formal methods, or between the phases of the software development cycle. Integration of tools and methods has often been put forward as a way to advance the capabilities of verification: finding a suitable semantics, for example, to enable the use of different methods in combination [27]. Some formal methods projects have attempted to include this kind of integration from the outset with varying levels of success.

Autofocus [38] developed in 1996, is one illustrative example of tool integration, providing links to methods such as SMV, formal proof tools and model checking, graphical description techniques and test methods. Other tools attempt to provide partial or full development cycle support. The B method [39] covers the complete software development cycle from specification, design, proof and code generation. Estelle [40], a formal specification standard developed in the 1970s, provides a compiler, simulator and debugger, a state/event table generator and test driver generator. LOTOS [41] provides a compiler, checker, IDE, and graphical interface.

Whereas model checking approaches to verification have had some major industrial successes [27], theorem provers have had a difficult time getting similar traction in the marketplace. Tool support and other attempts at assimilating and combining different provers have helped increase their usefulness. The SRI specification and verification toolset PVS [42] which includes a type checker, an interactive theorem prover, symbolic model checker, a code generator and a random tester, is probably one of the most popular theorem proving tools with industry and research institutes. Isabelle [43], a generic theorem prover from Cambridge University and Munich TU is distinguished by its flexibility: it can be configured to work with any logic eg. modal logic. Some logics such as FOL, HOL and ZF set theory are predefined. KIV [44] includes an interactive theorem prover which they claim, can automatically find 80-100% of the proof steps using a number of heuristics. This is supported by a powerful graphical user interface, and automatic generation of counter examples, to assist in interpreting failed proofs.

Researchers continue to search for ways to optimize the usefulness of theorem prover technology by configuring and combining them. Schumann in [45] describes how to control provers for optimal performance using different search paradigms, parallel execution, counterexamples etc. He demonstrates how automatic provers can be
applied successfully in key software engineering areas like security and logic-based software reuse.

Rushby proposes a framework for integrating methods in [30], which he calls the 'Evidential Tool Bus'. “It provides a way to loosely integrate verification components so that they can collaborate to solve problems beyond the capability of any single component”. Prover Technology products, such as Prover Plug_In [46], are one attempt to provide this kind of loose integration to tool developers, including some of the tool development companies mentioned in this paper. Prover Technology’s products are essentially software components and include proof engines for combinational logic, model checking, and data proving, which can be assimilated into software development tools.

SCALABILITY AND VERSATILITY

Scaleability here is intended to mean the ability of a formal method to accurately specify, verify and implement an industry-scale system. Versatility refers to the ability of formal methods to deal with industry-scale systems which have concurrent processes, temporal constraints, real-time constraints and hybrid system behavior. There exist a wide selection of specification languages, both model-oriented (Z,VDM) and property-oriented (OBJ) which have industry-scale capabilities, although debates persists as to the relative merits of each [47]. Scaleability has tended to be more problematic with system verification than with system specification. A perennial complaint about formal verification is that it may be possible to thoroughly verify a system, but only when the scale is minimised, or when inordinate amounts of time and cost are justifiable.

As current model checkers become capable of state counts of more than $10^{120}$ [48], reachability is less of a problem than before, but this technique is not always suitable. Proof based techniques, the alternative verification approach, have generally been less successful. Most automated theorem provers fail to scale efficiently, whereas interactive theorem provers have tended to be more scaleable. However, interactive provers are not attractive from the software engineer’s point of view, since they require a high level of mathematical expertise to be useful.

The verification of safety aspects of the Paris driverless Metro using the B Method is probably the most cited example of system safety verification with automated theorem proving. The project was within budget and delay parameters [49], which was a major success story for formal methods. However, the adoption of the B Method has not been as widespread as was hoped.

Praxis, who have been in the business of delivering formally correct systems for decades, has fully verified small to medium industry-scale systems (to date largest scale projects are ~200K loc). Other industry-scale formal methods successes have focused on niche needs such as concurrency [50, 51], real-time constraints and continuous/discrete variable systems. Model checking tools built on the basis of timed automata research include UPPAAL [52] and KRONOS [53]. Timed model checkers have been successfully used to verify many industry-scale system properties such as guaranteeing timeliness of multimedia data packets [54], the correctness of an automobile gear controller [55], the timeliness of a time-triggered networking protocol [56] and the correctness and timeliness of a real-time scheduler [57].

Similarly, hybrid automaton model checking has been widely used for hybrid system verification [58] and model checking has been shown to be decidable for important classes of hybrid systems [59], [60]. Model checking tools built on the basis of this research include Hytech [61], Verishift [62], and CheckMate [63]. These tools and techniques have been applied in many case studies, for example, hybrid control design for a wheeled mobile robot; modelling and control of SMT manufacturing lines using hybrid dynamic systems; hybrid control of an automotive robotized gearbox for reduction of consumption and emissions; safety verification of model helicopter controller using hybrid I/O automata; adaptive cruise control; and application of hybrid control to CPU reservations [64].

COST EFFECTIVENESS

Software development projects in general have been notoriously poor at demonstrating measured cost effectiveness. Formal methods and tools have not generally been any better in this regard, yet measured cost effectiveness is probably the single most convincing criterion in the adoption of such tools by industry. Exceptions to this are B Method projects which provide metrics for B model size, number of ADA lines and number of proofs [65]. Similarly, projects by both Praxis and Perfect Developer [36] provide metrics on number of specification lines, lines of programming code, number of verification conditions, number of valid conditions proved, processing time per condition, etc. Unit measures of work involved in applying formal methods are essential in order to calculate reliable costs and benefits of future projects.

PARTIAL USAGE

Good decisions made early in a software development cycle tend to have a positive cumulative effect throughout the life-cycle. Accurate requirements and specifications tend to uncover flaws and misconceptions early, helping to control project risks, and avoid more costly correction later. It follows than usage of formal methods in the early part of the development cycle will tend to be beneficial, even if later refinement and implementation is carried out without formal methods. An extension of this idea is the partial usage of formal methods to implement specific programs of a system or refactoring critical sections of legacy systems.
Some formal method projects explicitly aim from the outset to facilitate partial usage, such as the common framework for algebraic specification CASL [66] and CSP [51].

Many case-studies using the LOTOS toolset, CADP, since 1990, listed in [67], conclude that significant benefit was found in usage during specification and design.

Behavioral Interface Specification Languages (BISL) are a type of formal method which is very conducive to partial usage. A BISL describes two important aspects of a programming module:

- interface, i.e. names and static information
- behavior of the module

"BISLs are inherently language-specific because they describe interface details for clients written in a specific programming language" [68]. JML specifications, for example, are written as comments or annotations in Java program files, or alternatively they can be in written separate files. Either way, this specification code will not be compiled. Extended static checkers (ESC) such as ESC/Java [69] [70], use a subset of a specific programming language, to help detect bugs in the code. ESC/Java is very conducive to a gradual approach to adoption of formal methods, but specific to development in Java. Although it uses sophisticated automatic theorem proving it does not set out to verify properties of object oriented classes, but rather to finding common program errors at compile-time. Because the verification features are represented by annotations in the programming language, it may be more conducive to gradual adoption by software programmers. This might lead, for example, to the use of PVS for verification tasks outside the capabilities of an ESC. Annotations also facilitate the investment of formal methods in the critical sections of programs only, and refactoring legacy code.

RECOGNISED QUALITY STANDARDS

Standards which deal with the quality of software, such as IEC 61508, offer the software engineer guidelines and recommendations. When combined with the independent assessment of an external expert or certifier, they provide the customer with a greater level of assurance about the quality of the product. Similarly, the combination of adopting a formal method which is a recognized standard or a recognized best of breed, with external assessment or certification provides the customer assurance that due care has been taken with the safety of the software.

There are ISO standards for LOTOS, Estelle, VDM and Z. The B Method is a recognized leader in the field. CASL [66], the common framework for algebraic specification may become reference point for best practice in time. Others are recognised in specific domains. Esterel’s SCADE, for example, is the standard for the development of safety-critical embedded software in the avionics industry. HOL, PVS and Isabelle are frequently cited in research and industry projects as the preferred theorem provers.

APPLICABILITY OF THE CLASSIFICATION

The foregoing exposition demonstrated the relevance of this classification to the trade-offs that software engineers frequently need to manage. In the course of the review of various methods, the classification was applied to over forty specific tools and methods and summarized using the following simplistic mechanism for quantifying the evidence for these criteria:

- Significant evidence = 2
- Some evidence = 1
- Little or no evidence = 0

Figure 4 shows how the mechanism could be used. Out of over thirty methods, six or seven stand out clearly, helping to justify the selection of particular methods which are conducive to the goals of a project.

Each method is listed in the left-hand column and a score (0-2) is given to each of the 6 criteria for each method. Additionally, a weighting of 1.5 was applied to the criteria for Scaleability, Cost and Recognised Standards. The scoring and weighting mechanism chosen would depend on the needs of the project or business.
This kind of classification and scoring can help make business sense of formal methods, facilitating the organization and communication of thinking and decision-making about their suitability for a particular purpose.

Real-time, hybrid methods, for example, have clear application to the embedded distributed environment of automotive software, yet Interval Temporal Logic, Duration Calculus and Hytech score very badly. Even UPPAAL scores only marginally well. This could guide decision-makers to invest in greater collaboration with researchers in many cases. One of the reasons these methods score badly is because of limited scalability. Even with investment, some problems will be too large to verify or they will not be verifiable in any practical timescale. The problem area should always be selected carefully and limited to specific well-defined in behavioral properties.

The fact that most of the top eight methods do not feature hugely in automotive software development may indicate that the control mechanisms for costing software components should be reviewed. Such formal methods may be more cost justifiable if the costs of software development, maintenance and the failure to reuse software components are taken into account.

One of the weaknesses of this classification is that it is entirely dependent on the accuracy, consistency and resources available to the classifier. The scores shown in Figure 4, for example, are based on information found in the referenced sources during the course of this review. However, given the limited resources for the review, these scores may not always be indicative of how particular methods would rate if an exhaustive study were carried out.

**CONCLUSION**

There is an increasing demand for safety in automotive vehicles. Some of software and electronic components increasingly used in vehicle production support safety-critical and high-reliability functions. Software engineering best practices including formal methods will need to be increasingly employed in the automotive industry in response to increasing safety requirements.

This paper has argued that one of the obstacles to the adoption of formal methods by the automotive industry is the lack of a common classification framework which connects business criteria to specific methods.

A classification method for formal methods was introduced, based on business criteria such as scalability, commercialization, cost effectiveness etc. Over forty specific methods and tools were review in terms of these criteria, and an exposition of the criteria given using some of the best methods in each case. The findings of this review were tabulated and presented for visual and metrical evaluation.

The scoring exercise helped to organize thinking and evaluation of formal methods in the context of the automotive industry. It supports the argument that cost control mechanisms, for example, may be part of the obstacle to the adoption of existing and highly successful methods.

It also lends support to the proposal that automotive engineering challenges involving both analytical and computational models, might lead the industry to invest in greater collaboration with researchers.
One of the future research directions arising from this paper would be an extensive survey to critically evaluate the application of the classification.

ACKNOWLEDGMENTS

This work is partially supported by Science Foundation Ireland under grant no. 03/CE2/I303_1.

REFERENCES

42. SRI, PVS http://pvs.csl.sri.com/
43. ISABELLE, http://www.cl.cam.ac.uk/research/hvg/Isabelle/
44. KIV, http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/.
   http://www.inrialpes.fr/vasy/pub/cadp/case-studies/


   http://secure.ucd.ie/products/opensource/ESCJava2/

70. ESC/JAVA2,
   http://secure.ucd.ie/products/opensource/ESCJava2/

**CATEGORY**

Safety-Critical Systems

**CONTACT**

Lero – The Irish Software Engineering Research Centre
University of Limerick, Limerick, Ireland
Phone: +353 61 20-3637
Fax: +353 61 21-3036
Email: pat.mcelligott@lero.ie