# SEVEN DEADLY SINS

Michael G. Hinchey[1] and Jonathan P. Bowen[2]

## Prologue (προλογος)

The use of formal methods is fraught with difficulties, any one of which could cause the downfall of a project depending on their use. We enumerate a number of pitfalls which should be avoided in order to help make sure a formal methods project is successful. While this cannot ensure favourable results, it will help to avoid failure, which is all too easy an outcome.

## Seven Deadly Sins

### 1. Epideictic (επιδεικτικος) *used for effect*

Formal methods should not be applied merely as a means of demonstrating one's ability, nor to satisfy company whim, or as a result of peer pressure. Just as the advent of 'object-orientation' saw many firms incur needless expense as they unnecessarily converted their systems to object-oriented implementations, there is a possibility that some could unnecessarily adopt formal methods. Realistically, the first thing that must be determined before a formal development is undertaken is that one *really* does need to use formal methods – whether it is for increased confidence in the system, to satisfy a particular standard required by procurers, or to aid in conquering complexity, etc.

There are, however, occasions where formal methods are not only desirable, but positively required. A number of standards bodies have not only used formal specification languages in making their own standards unambiguous, but have strongly recommended and in the future may mandate the use of formal methods in certain classes of applications [1,2].

### 2. Hyperbole (υπερβλη) *exaggeration*

Formal methods are not a panacea; they are just one of a number of techniques that, when applied correctly, have been demonstrated to result in systems of the highest integrity, and one should not dismiss other methods entirely [3,4]. Formal methods are no guarantee of correctness; they are applied by humans, who are obviously prone to error.

Various support tools such as specification editors, type-checkers, consistency checkers and proof checkers should indeed reduce the likelihood of human error ... but not eliminate it. System development is a human activity, and always will be. Software engineering will always be subject to human whim, indecision, the ambiguity of natural language, and simple carelessness.

### 3. Pistic (πιστικος) *too trusting*

One must be careful not to place too much trust in techniques and tools that have not been demonstrated to be 'correct'. While the use of formal methods should certainly reduce the number of anomalies in system devel-

[1]Computer Laboratory, University of Cambridge, UK & Dept. of Computer & Information Science, NJIT, USA
[2]Oxford University Computing Laboratory, UK

opment, inexperience and carelessness can result in specifications that reduce to *false*, and are as ambiguous as natural language specifications.

Similarly, no proof should be taken as definitive. Hand-proofs are notorious in not only admitting errors as one moves from one line to the next, but also at making gigantic leaps which are unfounded. From experience of machine checking proofs, most theorems are correct, but most hand proofs are wrong [6]. Even the use of a proof checker does not guarantee the correctness of a proof. While it does aid in highlighting unsubstantiated jumps, and avoidable errors, proof should never be considered an alternative to testing, although it may reduce the required amount of testing, and in particular unit testing.

**4. Oligarchy** (*ολιγαρχια*) *rule by a few*

The formal methods community tends to be very introspective. However, there has been considerable investment in existing software development techniques and it would be foolhardy to replace these *en masse* with formal methods. Instead it is desirable to integrate formal methods into the design process in a cost-effective manner. One way to do this is to investigate how an existing formal method can be combined effectively with an existing structured method already in use within industry.

In addition, novice formal developers must ensure that they have access to expert advice. The majority of successful formal methods projects to date have had access to at least one consultant who is already expert in the use of formal techniques. It appears to be very difficult to learn to use formal methods successfully without such help until sufficient local expertise has been built up to make this unnecessary.

**5. Ephemeral** (*εφημερος*) *transitory*

The application of formal methods to the development process should not be seen as 're-inventing the wheel'. Rather, exploiting reuse in formal development can (at least in theory) aid in offsetting some of the set-up costs (e.g., tools, training and education) of the development.

Formal methods (or formal specification languages, specifically) provide a means of unambiguously stating the requirements of a system, or of a system component. In this way, formally specified system components that meet the requirements of components of the new system can easily be identified. Thus components that have been formally specified *and sufficiently well documented* can be identified, reused and combined to form components of the new system.

It is important however not just to focus on the reuse of code that has been developed using a formal approach, but rather to reuse the formal specifications themselves also.

**6. Epexegesis** (*επεξηγησις*) *additional words*

Formal specifications *can* be made intelligible and acceptable to non-technical personnel [7], but only provided that they are augmented with sufficient amounts of informal explanation, diagrams, etc.

The formal specifier must not use a formal specification as a means of demonstrating his/her own ability, but rather must be willing to rework specifications to make them more intelligible to others. Similarly, abstraction should be used to hide unnecessary detail, making a specification suitable for different audiences.

**7. Maiandros** (*Μαιανδρος*) *meandering*

The use of formal methods radically changes the cost structure of system development [3]. Inexperience with cost-estimation has resulted in a number of formal developments running considerably over-budget and behind schedule. However, a large number of projects have come in on-time and within budget (see, for example, those projects reported in [8]).

Moreover, most of the successful applications of formal methods have employed formal methods from the early stages of development, replacing natural language and informal specification. Indeed, most projects have reported greatest savings at these early stages, where formal specification techniques have highlighted errors that could be corrected before the programming phase.

**Epilogue** (επιλογος)

Some of the words in the section headings above may be unfamiliar to some readers. The same problem occurs when formal methods are used. In the past, the necessary grounding for the use of formal methods has not been taught adequately on computer science courses, meaning that many software engineers, and even more managers, shy away from using formal methods because they feel they are out of their depth. Fortunately many undergraduate courses (at least in Europe, and particularly in the UK), do now teach the requisite foundations for using formal methods. However, the teaching is often rather unintegrated with the rest of the syllabus. More coordinated courses (e.g., see [5]) in which the mathematical foundations are subsequently applied to practical problems will help to produce more professional software engineers for the future.

**Acknowledgements**

Mike Hinchey is funded by ICL. Jonathan Bowen is funded by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant no. GR/J15186.

**References**

1. J.P. Bowen and M.G. Hinchey, Formal Methods and Safety-Critical Standards. *IEEE Computer*, 27(8): 68–71, August 1994.
2. J.P. Bowen and V. Stavridou, Safety-Critical Systems, Formal Methods and Standards. *IEE/BCS Software Engineering Journal*, **8**(4):189–209, July 1993. Revised version in J.P. Bowen (ed.), *Towards Verified Systems*, Elsevier Science, Real-Time Safety Critical Systems series, Volume 2, Chapter 1, pp 3–33, 1994.
3. J.P. Bowen and M.G. Hinchey, Seven More Myths of Formal Methods: Dispelling Industrial Prejudices. In T. Denvir, M. Naftalin and M. Bertran (eds), *FME'94: Industrial Benefit of Formal Methods*, Springer-Verlag, LNCS, 1994. Longer version available as University of Cambridge Computer Laboratory Technical Report No. 357, December 1994. Also to appear in *IEEE Software*, July 1995.
4. J.P. Bowen and M.G. Hinchey, Ten Commandments of Formal Methods. *IEEE Computer*, 28(4):56–63, April 1995. Also available as University of Cambridge Computer Laboratory Technical Report No. 350, September 1994.
5. D. Garlan, Integrating Formal Methods into a Professional Master of Software Engineering Program. In J.P. Bowen and J.A. Hall (eds), *Z User Workshop, Cambridge 1994*, Springer-Verlag, Workshops in Computing, pp 71–85, 1994.
6. J. Rushby, *Formal Methods and their Role in the Certification of Critical Systems*, Technical Report SRI-CSL-95-1, SRI International, Menlo Park, CA, USA, March 1995. Shorter version of Technical Report SRI-CSL-93-7, November 1993.
7. J.A. Hall, Seven Myths of Formal Methods. *IEEE Software*, **7**(5):11–19, September 1990.
8. M.G. Hinchey and J.P. Bowen (eds), *Applications of Formal Methods*. Prentice Hall International Series in Computer Science, 1995. In Press.