Formal Methods in Collaborative Projects

Anna Zamansky[1], Guillermo Rodriguez-Navas[2], Mark Adams[3] and Maria Spichkova[4]
[1]*Information Systems Department, University of Haifa, Carmel Mountain, 31905, Haifa, Israel*
[2]*School of Innovation, Design and Engineering, Malardalen University, Högskoleplan 1, 72218 Västeras, Sweden*
[3]*Proof Technologies Ltd, 119 Comer Rd, Worcester WR2 5JD, U.K.*
[4]*School of Science, RMIT University, 414-418 Swanston Street, 3001, Melbourne, Australia*

Keywords: Formal Methods, Collaboration, Industrial Applications.

Abstract: In this paper we address particular aspects of integration of formal methods in large-scale industrial projects, namely collaborative aspects. We review recent works addressing such aspects, identify some current trends and discuss directions for further research.

1 INTRODUCTION

The discussion on the feasibility and usefulness of formal methods (FM) in practice has been going on for over four decades. For all this time formal methods have been perceived as "difficult, expensive and not widely useful" despite their widely acknowledged advantages. Their understandability, comprehensibility and scalability have been hypothesized as hindering factors for FM adoption in industry. In "40 years of formal methods" (Bjørner and Havelund, 2014) admit that despite numerous books, publications, conferences and user groups, the gap between academic research in FM and its integration in large industrial projects is yet to be bridged. Moreover, despite that fact that FM are encouraged by safety standards, this encouragement does not provide any concrete methodology for application of FM. And yet, as stated by Abrial, adopting formal methods in a software company is more a strategical and methodological issue than a technical one (Abrial et al., 1991).

This leads to the consideration of an important aspect of integration of FM in industrial projects, which has received less attention thus far. Development of complex software products is a *collaborative* effort. Collaboration in software engineering is usually understood as artifact-based or model-based collaboration, where the focus of activity is on the production of new models, the creation of shared meaning around the models, and elimintation of error and ambiguity within the models (Whitehead et al., 2010). Collaboration involves communication of many different stakeholders, as well as the use of different descriptions with a large range of levels of formality. This leads to the natural question what is the role of FM with respect to collaboration? Does collaboration make applying FM more difficult? Are there implicit collaboration models in the existing FM? Is the lack of support of collaborative aspects of FM an additional hindrance to the adoption of FM in industry? How can FM support these interactions?

The aim of this paper is to make a first step towards answering the above questions by highlighting collaborative aspects of FM in large-scale industrial software projects. We start by briefly reviewing some recent large-scale industrial projects in which FM have been successfully applied. Then we discuss some particular aspects of large-scale projects which are directly related to collaboration. We finish with a summary, highlighting also potential directions for future research.

2 FORMAL METHODS IN LARGE-SCALE PROJECTS

Industrial adoption of formal methods (or lack of thereof) has been a major concern in the FM community for decades and numerous works have been written on the topic. Several papers in the nineties discussed common myths of FM (Hall, 1990; Bowen and Hinchey, 1995a) and FM commandments (Bowen and Hinchey, 1995b; Bowen and Hinchey, 2006; Bowen and Hinchey, 2005). Numerous surveys on industrial use of FM have been published, among the most recent ones (Craigen et al., 1993; Bloomfield et al., 2000; Woodcock et al., 2009).

Understandability and comprehensibility of FM have been hypothesized for years as hindering factors of industry takeup of FM. Interestingly, empirical studies show no decisive evidence for this hypothesis. (Zimmerman et al., 2002) investigated how various factors of state-based requirements specification language design affect readability using aerospace applications: no decisive conclusions were reached. (Snook and Harrison, 2001) presents an empirical study of practitioners' views related to understandability and usability of formal methods. In contrast to popular opinions, all the experienced interviewees agreed that typical software engineers have no real difficulties with understanding formal notations.

The more difficult thing, in most opinions, was finding useful abstractions from which models can be created. (Andronick et al., 2012; Fitzgerald et al., 1995) also agree that FM capabilities is readily acquired by technical experts; in particular, it may be easier to train domain experts in formal methods than formal methods practitioners in the domain.

Scalability of FM in large-scale projects is considered another important concern. The challenges in managing large-scale proofs were discussed in (Bourke et al., 2012). The authors draw on experiences from two large-scale projects involving formal proofs: the pervasive system-level verification of Verisoft (Alkassar et al., 2009), and the operating system microkernel verification in the L4.verified project (Klein et al., 2009).

Lightweight formal methods (Jones et al., 1996; Jackson, 2001) seem a particularly promising approach in this respect. The idea is to concentrate just on the areas of biggest risk, and aim to establish just partial properties, or even just find errors missed by conventional software testing, rather than establish total absence of error. (Atzeni et al., 2014) report on experiences from applying model checking to the authentication system of a large scale security-oriented project. One of the lessons learnt is that lightweight verification does not necessarily require special experts and can be assigned to the testing team, who have an extensive knowledge of the system and prioritising errors. Similar insights are described in (Bennion and Habli, 2014), where model checking was applied to aero-engine monitoring software.

One of the recent successes in formal specification is reported in (Zave, 2012), where Alloy was applied to model and analyze a ring-maintenance protocol. A more expressive language, TLA^{+} has recently begun to be employed for checking correctness of Amazon Web Services (Newcombe et al., 2015). The use of a lightweight formal technique of Analytical Software Design is reported in (Osaiweran et al., 2015) report on the experiences with the embedding of ASD into the development processes.

Large-scale projects usually involve not a single representation level, but a stack of methods and tools, where the transition between them might be very challenging. Several approaches attempted to overcome this obstacle. For instance, the Ptolemy approach introduces a general way to combine heterogeneous models of embedded systems, cf. (Eker et al., 2003). To our best knowledge, the first work on achieving a *pervasive formal development process* for embedded applications starting with informal textual specification and leading to verified machine-code was done within the Verisoft-Automotive project (a part of Verisoft), cf. (Botaschanjan et al., 2008), covering the entire seamless pervasive development process. This project was done in collaboration between TU Munich, Saarland University and BMW Group.

The first steps towards a methodology for development of verified embedded system have been done in (Botaschanjan et al., 2006; Botaschanjan et al., 2005). For example, a typical setting found in the automotive domain, a time-triggered operating and communication bus system, has been verified (Kühnel and Spichkova, 2007). Earlier results of the Verisoft project have shown the methodology for later verification phases, in particular the relation between the application model and its execution environment, e.g. the operating system. The successor project, Verisoft-XT (Spichkova et al., 2012) was done in collaboration between TU Munich and Robert Bosch GmbH. Its core part developing a Cruise Control System with focus on system architecture and system verification. Verisoft-XT led to a development methodology for safety-critical systems with focus on the tool chain, which is employed in the design, implementation and verification phases of the methodology.

Another two projects on embedding formal and semi-formal methods in the automotive industrial development cycle were done in collaboration between TU Munich and DENSO Corporation, cf. (Feilkas et al., 2009; Feilkas et al., 2011). The core of both projects was elaboration of top-down methodologies for the development of automotive software systems and adapting them to the case studies provided by DENSO Corporation: an Adaptive Cruise Control system with Pre-Crash Safety functionality, and a keyless-entry system for smart vehicles.

3 EXISTING WORK ON COLLABORATIVE SOFTWARE DEVELOPMENT

Collaboration in software engineering is attracting increasing attention as projects relying on distributed development of software artifacts are becoming more popular. There is a wide recognition for the need of tools and methodologies to support collaborative development (De Jonge et al., 2001; Whitehead et al., 2010). The term *groupware* has been recently coined (Bani-Salameh and Jeffery, 2014) in order to refer to toolsets that support not only the technical aspects of distributed software engineering, but also the human and social activities inherent to any collaborative software project.

The relevance of groupware has been particularly recognized in the area of Global Software Engineering (GSE), in which development teams can be located in different countries, typically different continents, potentially raising cultural and communication difficulties that may have strong impact on the product quality and final cost. These challenges were reported in several surveys and systematic literature studies, which include discussions on existing tool support.

The most comprehensive review is probably (Portillo-Rodríguez et al., 2012), which reports a number of central activities for GSE: support to informal communication (e.g. screen sharing, videoconference, chat), visualization of project evolution, knowledge sharing (e.g. document sharing, requirements peer review, wiki) , distributed design (e.g. scenario creation, annotation of diagrams, CASE tool for UML models), distributed code review and code management. Other studies reporting similar results are (Lee et al., 2006) and (Trkman et al., 2013).

Interestingly, none of the consulted surveys makes any reference to any formal method or tool, and the two activities that are in principle more prone to formalization (knowledge sharing and distributed design) seem to be implemented at best with semi-formal approaches such as, for example, scenario creation and UML designs. Particularly, we noticed that the term "formal" is often used in contraposition to "colloquial". For instance, (Lee et al., 2006) encourages the use of formal communication, meaning either exchange of formal (i.e. written) text documents or formal meetings (i.e. with agenda and minutes) instead of informal communication (casual conversations). This observation is a good indicator of the little penetration that FM has had in this research area.

There are, however, some papers that report, or at least suggest, utilization of FM in the context of GSE. (Kuhrmann et al., 2013) suggests the use of software process metamodels (like SPEM) in order to specify the artifacts and data exchanged by each software process, thus abstracting away from local software processes and allowing composability and consistency analysis. (Cheng et al., 2009) proposes development of a new requirements vocabulary for self-adaptive systems that would introduce flexibility in the requirements specification, capturing uncertainty and making it easier for distributed implementation. (Zhiming et al., 2014) proposed an interesting scheme of modeling agents, which would monitor and assess the quality of the cooperation and assist the agents to find out better solutions, for instance reassigning roles.

4 COLLABORATIVE ASPECTS OF FORMAL METHODS

Given the relatively small interest that the application of FM has generated in this research field, there is an urgent need to address questions related to how social, cognitive, economic and managerial aspects of FM can be adapted to better support collaborative work. In particular,

- The use of FM in a collaborative project implies communication between an FM expert (or team of experts) with other stakeholders with different backgrounds. How can such communication be supported and improved?
- FM applications involve tasks and artifacts different than those in traditional software engineering. How can we evaluate and increase productivity, and what tool support is needed?
- Management of FM projects plays a crucial role in bridging the gap between academic stand-alone projects and large-scale collaborative projects. What managerial factors should be addressed in the context of FM?

In what follows we briefly review existing works that are relevant to the points mentioned above.

4.1 Communication

Application of FM in collaborative projects requires an interplay between formal and informal methods, which use different levels of formality in descriptions. For example, in the SACEM project (Guiho and Hennebert, 1990) a difficulty was reported in in communication between the verifiers and other engineers, who were not familiar with the formal specification method. This had to be overcome by providing the engineers with a natural language description derived manually from the formal specification.

Informal descriptions are easier to understand by non-expert stakeholders, but they may give rise to ambiguity, incompleteness or inconsistency; formal ones are more difficult to learn and require extensive mathematical training, but they provide mathematical rigor. Semi-formal approaches using diagrammatic or other visual notations, such as UML, fall somewhere in the middle of this range. There are numerous works that focus on formalizing a given type of informal or semi-formal descriptions. In what follows we mention some approaches with a particular focus on improving communication between stakeholders with different backgrounds.

(Knight et al., 2001) addresses the need to integrate both natural language text and formal language descriptions during the process of specification and provides a toolset suitable for industrial development, in which natural language is viewed as an integral and essential part of a specification.

(Maier and Hess, 2014) highlights this need in the context of interaction design: formal models are crucial for analyzing elicited requirements and designing a product that satisfied these requirements. Non-formal methods, like an open requirements elicitation in form of workshops or interviews may fail gaining all necessary information if there is no formal model which builds the basis for the non-formal methods. In this work non-formal methods, based on a conceptual formal model are developed that are easy to apply by non-experts.

(Schuts and Hooman, 2015) describe another approach to improve communication between stakeholders by proposing a formal modelling approach for the concept phase, using a light-weight modelling tool to formalize system behaviour, decomposition and interfaces and evaluating it Philips HealthTech.

Industrial data show that about 50 percent of product defects result from flawed requirements and up to 80 percent of rework efforts can be traced back to requirement defects (Wiegers, 2001). It would be interesting to investigate how many of these requirement defects are in fact communication problems/mistakes, which can be potentially reduced by applying FM.

4.2 Productivity

Considerations of time- and cost-effectiveness of applying FM are crucial in large-scale collaborative projects. This leads to the need for *evaluation* and, ultimately, *reduction* of the cost of FM efforts in terms of time and money. (Andronick et al., 2012) points out the lack of good understanding of what to measure in projects using formal methods. (Jeffery et al., 2015) proposes a concrete research agenda on metrics, cost models and estimation methods for large-scale projects employing FM.

The study in (Staples et al., 2014) addresses the issue of proof productivity: data on size of proofs and human effort was extracted from the history of the development of nine projects associated with seL4 microkernel; lines of proof were found to be very highly correlated with human effort, the limitations of this measure are also discussed.

(Freitas and Whiteside, 2014) introduce proof patterns, which aimed to provide a common vocabulary for solving formal methods proof obligations by capturing and describing solutions to common patterns of proof. The aim is to enable less experiences proof engineers to identify common situations and tackle them using proof patterns.

Another issue which is directly related to increasing effectiveness of FM use in collaborative projects is *proof reuse*. Several techniques have been considered, such as incremental proof reuse (Beckert and Klebanov, 2004), global abstraction methods (Huang et al., 1994), constructive methods (such as ones used in KIV (Balser et al., 2000)), and similarity-based methods (Melis and Schairer, 1998).

4.3 Tool Support

For FM to be beneficiary for software development projects, they must be fully integrated into the software lifecycle. This means that appropriate tool support is a crucial factor. Lack of appropriate tool support was indeed hypothesized as one of the main obstacles for a wider adoptions of FM in industry. As noted in (Bjørner and Havelund, 2014), in their early usage the first formal specification languages, VDM and Z, did not provide even the simplest syntax checker, so that “the mere thought that three or more programmers need collaborate on code development occurred much too late in those circles”. Means are now finally being taken to remedy this situation. For instance, the VSTTE (Verified Software: Theories, Tools and Experiments) initiative aims to advance the state of the art in the science and technology of software verification through the interaction of theory development, tool evolution, and experimental validation.

Although empirical data shows that the majority of product defects result from problems traced to requirements (Wiegers, 2001), tool support for the specification phase has so far been scarce. Recent specification development tools are IBMs RuleBase PE (RuleBase, 2015), and the academic tool RAT (Bloem et al., 2007). These tools enable the designer to explore a specification's properties. (Knight et al., 2001) describes another toolset developed with the intent of providing comprehensive facilities for creating formal specifications in production software development.

A crucial lesson learnt in the verification experience of (Bourke et al., 2012) is the importance of proof automation, because it decreases the cognitive load on the analyst as well as shortening proof scripts and thus reducing maintenance costs. Both projects used Isabelle/HOL prover, which can be soundly extended with customised automation. Another finding is that automated support to help analysts find theorems to use in their proof scripts can greatly increase productivity.

A recent line of research of *human factors in FM* aims to support the human expert in applying FM, producing methodologies and tools for making FM more comprehensible, easy to use, readable, while hiding complexity and reducing errors.(Spichkova, 2013), e.g., introduced the idea of applying ideas from engineering psychology at the specification phase of system development, aiming to increase readability and reduce chances of errors. These ideas can be extended to the domain of *collaborative* FM by developing intuitive and easy-to-use technological platforms for collaborative speficiation and verification efforts.

4.4 Management

(Mandrioli, 2015) points out to too little attention to the managerial and political aspects as one of the main factors hindering adoption of FM in industry. Indeed, information on managerial factors of FM in collaborative projects is scarce. (Stidolph and Whitehead, 2003) provides guidelines in deciding whether a particular project is a good candidate for the use of FM and describes some of the management issues to be considered when running such projects.

(Andronick et al., 2012) reports on process and management aspects of a recent landmark formal verification project of the microkernel seL4. To the best of our knowledge, this is the only report to provide a full descriptive model of the verification process. The paper further stresses the need for decision-making tools for FM project management, which are currently not available.

5 SUMMARY

In this position paper we have reviewed some recent large-scale projects which applied FM, and discussed some important collaborative aspects of such projects. It is our hope that this paper will stimulate discussion on research agenda to better address collaborative aspects of FM, pointing out gaps that can be filled through collaboration between academy and industry.

ACKNOWLEDGEMENTS

This work was partially funded by the Israel Science Foundation under grant agreement no. 817/15 and by the Swedish Governmental Agency for Innovation Systems (VINNOVA) under project 2013-01299.

REFERENCES

Abrial, J.-R., Lee, M. K., Neilson, D., Scharbach, P., and Sørensen, I. H. (1991). The b-method. In *VDM'91 Formal Software Development Methods*, pages 398–405. Springer.

Alkassar, E., Hillebrand, M. A., Leinenbach, D. C., Schirmer, N. W., Starostin, A., and Tsyban, A. (2009). Balancing the load: Leveraging semantics stack for systems verification. *Journal of Automated Reasoning: Special Issue on Operating Systems Verification*, 42, Numbers 2-4:389–454.

Andronick, J., Jeffery, R., Klein, G., Kolanski, R., Staples, M., Zhang, H., and Zhu, L. (2012). Large-scale formal verification in practice: A process perspective. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 1002–1011. IEEE.

Atzeni, A., Su, T., and Montanaro, T. (2014). Lightweight formal verification in real world, a case study. In *Advanced Information Systems Engineering Workshops*, pages 335–342. Springer.

Balser, M., Reif, W., Schellhorn, G., Stenzel, K., and Thums, A. (2000). Formal system development with kiv. In *Fundamental approaches to software engineering*, pages 363–366. Springer.

Bani-Salameh, H. and Jeffery, C. (2014). Collaborative and social development environments: A literature review. *Int. J. Comput. Appl. Technol.*, 49(2):89–103.

Beckert, B. and Klebanov, V. (2004). Proof reuse for deductive program verification. In *Software Engineering and Formal Methods, 2004. SEFM 2004. Proceedings of the Second International Conference on*, pages 77–86. IEEE.

Bennion, M. and Habli, I. (2014). A candid industrial evaluation of formal software verification using model checking. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 175–184. ACM.

Bjørner, D. and Havelund, K. (2014). 40 years of formal methods. In *FM 2014: Formal Methods*, pages 42–61. Springer.

Bloem, R., Cavada, R., Pill, I., Roveri, M., and Tchaltsev, A. (2007). Rat: A tool for the formal analysis of requirements. In *Computer aided verification*, pages 263–267. Springer.

Bloomfield, R., Craigen, D., Koob, F., Ullmann, M., and Wittmann, S. (2000). Formal methods diffusion: Past lessons and future prospects. In *Computer Safety, Reliability and Security*, pages 211–226. Springer.

Botaschanjan, J., Broy, M., Gruler, A., Harhurin, A., Knapp, S., Kof, L., Paul, W., and Spichkova, M. (2008). On the correctness of upper layers of automotive systems. *Formal aspects of computing*, 20(6):637–662.

Botaschanjan, J., Gruler, A., Harhurin, A., Kof, L., Spichkova, M., and Trachtenherz, D. (2006). Towards Modularized Verification of Distributed Time-Triggered Systems. In *FM 2006: Formal Methods*, pages 163–178. Springer.

Botaschanjan, J., Kof, L., Kühnel, C., and Spichkova, M. (2005). Towards Verified Automotive Software. In *2nd International ICSE workshop on Software*. ACM.

Bourke, T., Daum, M., Klein, G., and Kolanski, R. (2012). Challenges and experiences in managing large-scale proofs. In *AISC/MKM/Calculemus*, pages 32–48.

Bowen, J. P. and Hinchey, M. G. (1995a). Seven more myths of formal methods. *IEEE software*, 12(4):34–41.

Bowen, J. P. and Hinchey, M. G. (1995b). Ten commandments of formal methods. *Computer*, 28(4):56–63.

Bowen, J. P. and Hinchey, M. G. (2005). Ten commandments revisited: a ten-year perspective on the industrial application of formal methods. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 8–16. ACM.

Bowen, J. P. and Hinchey, M. G. (2006). Ten commandments of formal methods... ten years later. *Computer*, 39(1):40–48.

Cheng, B. H., Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H. M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H. A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., and Whittle, J. (2009). Software engineering for self-adaptive systems: A research roadmap. In Cheng, B. H., Lemos, R., Giese, H., Inverardi, P., and Magee, J., editors, *Software Engineering for Self-Adaptive Systems*, pages 1–26. Springer-Verlag, Berlin, Heidelberg.

Craigen, D., Gerhart, S., and Ralston, T. (1993). An international survey of industrial applications of formal methods. In *Z User Workshop, London 1992*, pages 1–5. Springer.

De Jonge, M., Visser, E., and Visser, J. M. (2001). *Collaborative software development*. Citeseer.

Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. (2003). Taming heterogeneity - the ptolemy approach. In *Proceedings of the IEEE*, pages 127–144.

Feilkas, M., Fleischmann, A., Hölzl, F., Pfaller, C., Rittmann, S., Scheidemann, K., Spichkova, M., and Trachtenherz, D. (2009). A Top-Down Methodology for the Development of Automotive Software. Technical Report TUM-I0902, TU München.

Feilkas, M., Hlzl, F., Pfaller, C., Rittmann, S., Schtz, B., Schwitzer, W., Sitou, W., Spichkova, M., and Trachtenherz, D. (2011). A Refined Top-Down Methodology for the Development of Automotive Software Systems - The KeylessEntry-System Case Study. Technical Report TUM-I1103, TU München.

Fitzgerald, J. S., Larsen, P. G., and Larsen, P. (1995). Formal specification techniques in the commercial development process. In *Position Papers from the Workshop on Formal Methods Application in Software Engineering Practice, International Conference on Software Engineering (ICSE-17), Seattle*.

Freitas, L. and Whiteside, I. (2014). *Proof patterns for formal methods*. Springer.

Guiho, G. and Hennebert, C. (1990). Sacem software validation. In *Software Engineering, 1990. Proceedings., 12th International Conference on*, pages 186–191. IEEE.

Hall, A. (1990). Seven myths of formal methods. *Software, IEEE*, 7(5):11–19.

Huang, X., Kerber, M., Richts, J., and Sehn, A. (1994). Planning mathematical proofs with methods. *Elektronische Informationsverarbeitung und Kybernetik*, 30(5/6):277–291.

Jackson, D. (2001). Lightweight formal methods. In *FME 2001: Formal Methods for Increasing Software Productivity*, pages 1–1. Springer.

Jeffery, R., Staples, M., Andronick, J., Klein, G., and Murray, T. (2015). An empirical research agenda for understanding formal methods productivity. *Information and Software Technology*, 60:102–112.

Jones, C. B., Jackson, D., and Wing, J. (1996). Formal methods light. *Computer*, 28(4):20–22.

Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and Winwood, S. (2009). sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA. ACM.

Knight, J. C., Hanks, K. S., and Travis, S. R. (2001). Tool support for production use of formal techniques. In *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*, pages 242–251. IEEE.

Kühnel, C. and Spichkova, M. (2007). Fault-Tolerant Communication for Distributed Embedded Systems. In *Software Engineering and Fault Tolerance*, Series on Software Engineering and Knowledge Engineering.

Kuhrmann, M., Fernández, D. M., and Gröber, M. (2013). Towards artifact models as process interfaces in distributed software projects. In *Proceedings of the 2013 IEEE 8th International Conference on Global Software Engineering*, ICGSE '13, pages 11–20, Washington, DC, USA. IEEE Computer Society.

Lee, G., DeLone, W., and Espinosa, J. A. (2006). Ambidextrous coping strategies in globally distributed software development projects. *Commun. ACM*, 49(10):35–40.

Maier, A. and Hess, S. (2014). We need non-formal methods based on formal models in interaction design. In *Building Bridges: HCI, Visualization, and Non-formal Modeling*, pages 150–164. Springer.

Mandrioli, D. (2015). On the heroism of really pursuing formal methods. In *Formal Methods in Software Engineering (FormaliSE), 2015 IEEE/ACM 3rd FME Workshop on*, pages 1–5. IEEE.

Melis, E. and Schairer, A. (1998). Similarities and reuse of proofs in formal software verification. In *Advances in Case-Based Reasoning*, pages 76–87. Springer.

Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., and Deardeuff, M. (2015). How amazon web services uses formal methods. *Communications of the ACM*, 58(4):66–73.

Osaiweran, A., Schuts, M., Hooman, J., Groote, J. F., and van Rijnsoever, B. (2015). Evaluating the effect of a lightweight formal technique in industry. *International Journal on Software Tools for Technology Transfer*, pages 1–16.

Portillo-Rodríguez, J., Vizcaíno, A., Piattini, M., and Beecham, S. (2012). Tools used in global software engineering: A systematic mapping review. *Inf. Softw. Technol.*, 54(7):663–685.

RuleBase, I. (2015). Ibm rulebase homepage.

Schuts, M. and Hooman, J. (2015). Formalizing the concept phase of product development. In *FM 2015: Formal Methods*, pages 605–608. Springer.

Snook, C. and Harrison, R. (2001). Practitioners' views on the use of formal methods: an industrial survey by structured interview. *Information and Software Technology*, 43(4):275–283.

Spichkova, M. (2013). Design of formal languages and interfaces:formal does not mean unreadable. *IGI Global*.

Spichkova, M., Hölzl, F., and Trachtenherz, D. (2012). Verified System Development with the AutoFocus Tool Chain. In *2nd Workshop on Formal Methods in the Development of Software*, WS-FMDS.

Staples, M., Jeffery, R., Andronick, J., Murray, T., Klein, G., and Kolanski, R. (2014). Productivity for proof engineering. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 15. ACM.

Stidolph, D. C. and Whitehead, J. (2003). Managerial issues for the consideration and use of formal methods. In *FME 2003: Formal Methods*, pages 170–186. Springer.

Trkman, M., Vrhovec, S., Vavpoti?, D., and Krisper, M. (2013). Defending the need for a new global software approach: A literature review. In *Information Communication Technology Electronics Microelectronics (MIPRO), 2013 36th International Convention on*, pages 199–202.

Whitehead, J., Mistrík, I., Grundy, J., and van der Hoek, A. (2010). Collaborative software engineering: Concepts and techniques. In *Collaborative Software Engineering*, pages 1–30. Springer.

Wiegers, K. E. (2001). Inspecting requirements. *StickyMinds. com*.

Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Computing Surveys (CSUR)*, 41(4):19.

Zave, P. (2012). Using lightweight modeling to understand chord. *ACM SIGCOMM Computer Communication Review*, 42(2):49–57.

Zhiming, C., Zhe, Y., Menghan, W., and Jiangling, Y. (2014). The agents coordination and templates aggregation in distributed modeling. *International Journal of Hybrid Information Technology*, 7(2):369–378.

Zimmerman, M. K., Lundqvist, K., and Leveson, N. (2002). Investigating the readability of state-based formal requirements specification languages. In *Proceedings of the 24th International Conference on Software engineering*, pages 33–43. ACM.