An Empirical Research Agenda for Understanding Formal Methods Productivity

Ross Jeffery, Mark Staples, June Andronick, Gerwin Klein, Toby Murray
NICTA, Australia and
University of New South Wales, Australia
Level 4, 223 Anzac Parade
Kensington, NSW 2052 Australia
+61 2 9376 2000
<firstname.lastname>@nicta.com.au

ABSTRACT

Context: Formal methods, and particularly formal verification, is becoming more feasible to use in the engineering of large highly dependable software-based systems, but so far has had little rigorous empirical study. Its artefacts and activities are different to those of conventional software engineering, and the nature and drivers of productivity for formal methods are not yet understood.

Objective: To develop a research agenda for the empirical study of productivity in software projects using formal methods and in particular formal verification. To this end we aim to identify research questions about productivity in formal methods, and survey existing literature on these questions to establish face validity of these questions. And further we aim to identify metrics and data sources relevant to these questions.

Method: We define a space of GQM goals as an investigative framework, focusing on productivity from the perspective of managers of projects using formal methods. We then derive questions for these goals using Easterbrook et al.'s [14] taxonomy of research questions. To establish face validity, we document the literature to date that reflects on these questions and then explore possible metrics related to these questions. Extensive use is made of literature concerning the L4.verified project completed within NICTA, as it is one of the few projects to achieve code-level formal verification for a large-scale industrially deployed software system.

Results: We identify more than thirty research questions on the topic in need of investigation. These questions arise not just out of the new type of project context, but also because of the different artefacts and activities in formal methods projects. Prior literature supports the need for research on the questions in our catalogue, but as yet provides little evidence about them. Metrics are identified that would be needed to investigate the questions. Thus although it is obvious that at the highest level concepts such as size, effort, rework and so on are common to all software projects, in the case of formal methods, measurement at the micro level for these concepts will exhibit significant differences.

Conclusions: Empirical software engineering for formal methods is a large open research field. For the empirical software engineering community our paper provides a view into the entities and research questions in this domain. For the formal methods community we identify some of the benefits that empirical studies could bring to the effective management of large formal methods projects, and list some basic metrics and data sources that could support empirical studies. Understanding productivity is important in its own right for efficient software engineering practice, but can also support future research on cost-effectiveness of formal methods, and on the emerging field of Proof Engineering.

Categories and Subject Descriptors

Author's preprint of paper published in Information and Software Technology, Vol. 60, April 2015, pp. 102-112
http://dx.doi.org/10.1016/j.infsof.2014.11.005

D.2.4 [**Software Engineering**]: Software/Program Verification – *correctness proofs, formal methods*; D.2.8 [**Software Engineering**]: Metrics – *process metrics, product metrics*; D.2.9 [**Software Engineering**]: Management – *productivity.*

General Terms
Management, Measurement, Experimentation, Verification.

Keywords
Empirical Software Engineering, Productivity, GQM, Formal methods, Formal Verification, Proof Engineering.

1. INTRODUCTION

Formal methods is the mathematical specification, design and verification of computer systems. It can provide a much higher level of assurance than traditional code-and-test approaches to software engineering. Software engineering researchers have aspired to see the wide-spread use of formal methods since the 1970s, but only recently have technologies and techniques developed enough for it to become practical for use in non-trivial systems development projects. Increasingly, software systems are also safety- or security-critical and so could benefit from formal methods verification to provide direct evidence about system dependability. Nonetheless, to broaden its reach requires that its costs be better understood and where possible reduced. In particular, as noted by Klein [29], verification of low level implementations has been considered to be prohibitively expensive until recently. But this view is changing with the development of newer tools and methods. Studies such as those by King et al. [27] show the benefits of formal specification and verification. In this study they were able to show that for the SHOLIS system, the use of a Z proof of properties at the requirements and design levels was "substantially more efficient at finding faults than the most efficient testing phase". In addition they concluded that verification of SPARK Ada code was "more efficient at error detection than unit testing". However studies such as this are still rare and the insights provided into formal methods productivity partial and inconsistent. Thus we need a deeper understanding of productivity in this context.

Formal methods involves different kinds of development artefacts than traditional software engineering, and can provide qualitatively different kinds of assurance. As will be discussed in this paper, some of the traditional metrics used in empirical software engineering do not apply in projects using formal methods. New metrics will be needed. In a previous paper [1], we claimed that there is not yet a good understanding of what to measure in projects using formal methods. There is a need for research on metrics, cost models and estimation methods for such projects. This echoes much earlier statements [15] [48], which indicates the lack of progress in this field over many years.

In this paper we define a space of research questions about the productivity of formal methods, and define a collection of metrics that bear on these research questions. For each question, evidence provided in the literature pertaining to that question is shown. Our paper provides a research agenda and a call to researchers in Empirical Software Engineering to study formal methods projects, and for researchers and practitioners in formal methods to collaborate on the opportunity provided by these empirical studies.

The questions in this paper are relevant whether we want to (a) reduce the cost of formal methods, (b) have it scale better, or (c) provide evidence to compare its cost-effectiveness with conventional software engineering. For any of these goals we need to be able to characterize the

cost of formal methods and uncover appropriate metrics for this context. We can then use these metrics to develop an understanding of task size and effort drivers, and the empirical relationships between these and effort and schedule.

The existing literature is used in three ways in this paper. Firstly, we motivate the work using published papers showing the current lack of empirical evidence concerning cost, effort and quality of formal methods application in industry. Secondly we use literature to test the face validity of the research questions proposed, and finally we summarize literature that addresses guidelines and management issues concerning the application of formal methods. This finally provides a summary of the state of practice and a picture of the need for careful empirical research on the use and characteristics of formal methods in industry.

2. BACKGROUND

It is significant for this work that formal methods may use different activities to produce different artefacts as part of a different lifecycle compared with traditional software engineering. An illustration of this is shown in a formal verification process lifecycle model in Figure 1 below. In this model major artefact differences include the proofs themselves and the set of invariants of the system, which would not be present in conventional software engineering. The different activities that create and maintain those artefacts are also present in the process lifecycle.

Formal methods may target functional correctness, but this is not the only "quality" of software. Security properties and real-time performance may also be able to be supported by formal methods, but qualities such as maintainability or portability do not have accepted mathematical formalizations.

Many empirical studies of conventional software engineering have examined the impact of process or technology on defect density (the number of defects per code size). With conventional software engineering, defects may be able to be reduced, but no guarantees can be provided that all defects have been identified or eliminated. In contrast, formal methods offers the possibility to create software with zero defects. (Subject to the soundness of the logical reasoning used and the empirical validity of the models of specifications and machines.) This makes it qualitatively different to conventional software engineering. Klein [29] notes "formal methods held much promise in the 1970's but failed to deliver". However he further states "this situation has changed drastically" illustrating that formal verification of medium-level models of large systems of the order of 100,000 lines of C code has been demonstrated, and that formal verification of low-level implementations of systems around 10,000 lines of C code has also been achieved. The focus of Klein's paper [29] is on verification of operating systems, and it discusses in detail the Verisoft project and the L4.verified/seL4 project among others. What is revealed is that the methods, tools and skills available now make low-level verification a practical approach to be combined with testing and inspections in software development.

In their 2009 paper Woodcock et al. [50] find that "a weakness in the current situation is lack of a substantial body of technical and cost-benefit evidence from application of formal methods and verification technology". In summarizing their survey they report that 35% of respondents showed improvement in time and 12% a worsening. With regard to cost 37% reported an improvement and 7% a worsening, and with respect to quality, 92% reported an improvement and 8% reported worse quality. It is clear from this that a deeper empirical understanding is needed of the use of formal methods and their impact on cost, quality, and schedule. Woodcock et al. [50] also provide eight case study descriptions of relatively recent industrial projects using

formal methods. These are then used to observe that formal methods have "not seen widespread adoption…except arguably in the development of critical systems in certain domains".

In this paper we argue that given this, it is now time for the empirical software engineering community to add to our understanding of verification processes, artefacts, and contexts and to provide the necessary evidence needed to further the industrial application of formal verification in a cost effective manner. A first step in this is a deeper understanding of formal methods productivity. We focus intentionally on the use of formal methods in industrial scale projects as it is in this context that productivity becomes an issue and is additionally susceptible to empirical study.

The background literature is examined below as it relates to the research questions identified. A thorough review of the literature reveals little empirical study of the processes involved in the use of formal methods and inconsistent and incomplete findings to date concerning productivity in this domain.

3. RESEARCH GOALS

Our investigation uses the Goal Question Metric (GQM) approach [2][37]. GQM is a framework for the specification of a measurement system. It has three levels: a conceptual level of high-level measurement goals, an operational level of questions that bear on each goal, and a quantitative level of data and metrics that can be used to answer the defined questions. We first define a space of goals for empirical research on productivity in formal methods projects. In later sections we identify related research questions, and a range of supporting metrics.

In GQM, goals are defined according to a template [37]:

> to analyze an <object of study>
> in order to <purpose>
> with respect to <focus>
> from the point of view of <stakeholder>
> in the context of <environment>.

We fix two of these: the focus, and stakeholder. We consider a generic range of purposes, and consider three kinds of project environments for different kinds of formal methods projects, each with a corresponding collection of objects of study. These formal methods projects prove properties of requirements specifications and may include verification of design and/or code with respect to those requirements.

3.1 Stakeholder and Focus

We consider the point of view of one GQM *stakeholder*: managers of formal methods projects. We assume that their primary concerns are related to the classic iron triangle of project management: scope, duration, and effort (and the unstated factor - quality). However, as a simplifying assumption we take scope to be predefined for the project, and so take as our GQM *focus* the issue of *productivity*. We have chosen productivity as a focus because we believe that the key to broadening the massive potential impact of formal methods on software practice is to reduce its cost of application.

This does not imply, however, that the metrics derived in our study will not be of use in relation to other goals and questions that exist in regards to formal verification. As has been shown in the past [47] many goals may exist which then may lead to overlapping relevant questions and metrics. Thus although in this paper we focus on productivity in the use of formal methods in

industry, we do not assume that other issues such as fitness for purpose, effectiveness, user satisfaction and so on, will not be important in other empirical studies of productivity.

3.2 Purpose

Software is developed for a broad purpose: typically to support an organizational, social or business strategy. In a specific organization, a pure GQM approach would start with goals related to that specific purpose. However, in this research paper we instead use the generic goals for the management of formal methods projects given in SEI's *Goal-Driven Software Measurement* method [39]. This method defines four general reasons for measurement of entities (such as processes, products, or resources): to *characterize* (understanding and baselining entities), to *evaluate* (comparing entities with respect to a plan or standard), to *predict* (understanding relationships between entities to support planning), or to *improve* (identifying and tracking opportunities to change entities for the better).

The SEI generic reasons are similar to the five types of information systems theory proposed by Gregor [21]: *analysis* (say what is), *explanation* (say what is with how, when, why and where), *prediction* (what is and what will be), *explanation and prediction* (testable propositions and causal explanations), and *design and action* (how to do something). The five types of theory are arranged in a simple lattice. The research questions we define could also map into Gregor's meta-theory, but we use the SEI framework because it is a simpler partition of possible goals.

3.3 Context and Objects of Study

In this paper we consider three kinds of formal methods projects, involving progressively more formally defined artefacts and associated proofs. These three increasingly narrow GQM environment project contexts are:

1. Projects doing formal specification, i.e. developing formal *specifications* and *proofs* about the *specifications' properties*;
2. Projects performing requirements-driven design verification, i.e. developing *specifications*, developing *specification-level proofs* about the *specifications' properties*, developing a *design*, developing *refinement proofs* between the specifications and the design, and optionally developing direct *design-level proofs* about the *design's properties*; and
3. Projects performing requirements-driven code verification, i.e. developing *specifications*, developing *specification-level proofs* about the *specifications' properties*, developing a *design*, developing *high-level refinement proofs* between the specifications and the design, optionally developing direct *design-level proofs* about the *design's properties*, developing the system *code*, developing *low-level refinement proofs* between the design and the code, and optionally developing direct *code-level proofs* about the *code's properties*.

Code-level formal verification had been thought, in general, to be unfeasibly expensive for industry [22], but recent efforts including the L4.verified project [30] have demonstrated that it is becoming possible to verify strong properties of highly complex software. All of the project types above are requirements-driven – they create direct evidence [25] that a software system satisfies its requirements by delving ever-deeper into the design hierarchy. The refinement proofs listed above allow specifications' properties to be carried through to the design or code. Other kinds of projects using formal methods are possible. For example, a project may not have formal specifications or designs, but may still use formal methods techniques to prove properties

directly about code. We do not explicitly address such projects in this paper, although many of the research questions we identify may still apply to them.

Each of the three GQM environments above has a corresponding collection of GQM *objects of study*, which are formal verification artefacts, and the activities of creating and maintaining them:

1. Requirements specification, properties of the specification, and proofs of those properties;
2. Artifacts from 1, plus: designs, refinement proofs between requirements and designs, and optionally properties of the design and direct proofs of those properties; and
3. Artifacts from 2, plus: code, refinement proofs between the design and the code, and optionally properties of the code and proofs of those properties.

3.4 Summary: GQM Goal Space

Thus in terms of the GQM template [37], our space of goals is:

- to analyze **the activities and artefacts of formal methods**
- in order to **characterize/evaluate/predict/improve**
- with respect to **productivity**
- from the point of **view of managers of projects using formal methods**
- in the context of **projects that prove properties of formal specifications or which also formally verify the refinement of those specifications to design or code**.

4. RESEARCH QUESTIONS

For the space of GQM goals given above, we identify a list of research questions. This will not be an exhaustive list, but we use a theoretical categorization to systematically cover a range of question types. We then establish the face validity of the research questions by showing how existing literature identifies or provides evidence about them. Although our paper is not a systematic mapping study [41], our work also serves as a survey of the field.

We use the categories of research questions provided by Easterbrook et al. [14]. The four categories (and sub-categories) are based on the work of Meltzoff [36], and are as follows:

A. **Exploratory**: Existence; Description and classification; and Descriptive-comparative
B. **Base rate**: Frequency and distribution; Descriptive-process; Relationship
C. **Knowledge**: Causality; Causality-comparative; Causality-comparative interaction
D. **Design**

The first three categories (exploratory, base rate, and knowledge) are referred to by Easterbrook et al. as *deriving* knowledge. The focus of the *design* category is on developing better methods or tools to achieve software engineering goals that are based on, or utilize, knowledge gained from the first three categories of research questions. An example from conventional software engineering might include the use of techniques found to be statistically-significant for defect-detection in the proposal of new software inspection methods. Easterbrook et al. observe that software engineering research involves both knowledge and design questions.

We claim that SEI's generic reasons [39] can be mapped to Easterbrook's [14] question sub-categories as follows:

- characterize: existence; description and classification; descriptive process
- evaluate: descriptive-comparative
- predict: frequency and distribution; relationship

- improve: causality; causality-comparative; causality-comparative interaction; design

For each of these categories, we consider parameters of our GQM template, and identify specific research questions. So by covering these categories, we cover the generic goals. Thus we systematically derive research questions using these previously-published frameworks. However, we do not claim that we have derived a complete list of research questions for the area.

4.1 Exploratory Questions

Exploratory questions lead to qualitative knowledge useful for building tentative theories, understanding phenomena and identifying distinctions. Easterbrook et al. [14] discuss three types: Existence questions; Description and Classification questions; and Descriptive-Comparative questions.

4.1.1 Existence Questions

Easterbrook et al. [14] say that Existence questions are of the form "Does X exist?" This may seem like a trite question, but the use of formal verification is not yet common, and many software engineers and researchers will not yet have encountered formal verification in practice, so there may be reasonable doubt about this issue. Possible Existence questions are:

1. Are there instances of the formal methods project contexts in existence? [1], [7], [12], [18], [46]
2. Do the artefacts and activities exist in practice in these projects? [1], [46]
3. Is productivity an open issue in these projects? [1], [46], [44], [45]

Van Lamsweerde [46] notes that "the number of success stories in using formal specifications for real systems is steadily growing." He references a number of cases in transportation, power, telecommunications and security and notes evidence of higher quality and even lower cost. Woodcock et al. [50] say that "industrial use of formal methods for software has only performed formal specification, rarely formal verification". However Ghezzi et al. [19] observe that "formal verification of large production software is now increasingly feasible". The code-level formal verification of seL4 [30] demonstrates this. The uncertain impact of formal methods on productivity is a management issue [45].

There have been a number of individual experience reports [1][7][12][18] about the use of formal methods in industry, many of which discuss productivity and cost questions. Staples et al. [44] discuss the very limited empirical data available on formal methods projects, particularly concerning effort estimation which requires an understanding of size/effort relationships and factors which influence productivity in the estimation context.

The literature reveals that the use of formal methods appears to be increasing, particularly in the case of specification, but empirical evidence is sparse, and productivity is certainly an open issue. This is not unlike the evidence gathered in the papers by Petre [42] and Gorschek et al. [20]. In these two empirical studies it is revealed, among other details, that UML is not used extensively in industry and in the minority of cases where it is used it is informal and without tool support. Petre points out that these empirical findings are in conflict with the accepted view of the use of UML in industry. Thus these studies show the need to empirically question the assumptions made regarding the use of technologies in industry and this will apply to formal methods as much as any other tool/approach in software engineering. We must ask, are formal methods used in industry and if so, how are they used and what is their impact and characteristics?

4.1.2 Description and Classification Questions

Easterbrook et al. [14] say that Description and Classification questions are of the form "What is X like?", "What are its properties?", "How can it be categorized?", "How can we measure it?", "What is its purpose?", "What are its components?", "How do the components relate to one another?", and "What are all the types of X?" These questions provide high-level understanding of the kinds and attributes of units of analysis. Possible Description and Classification questions include:

1. What are the properties of formal methods artefacts and activities? [38] [44] [7]
2. How can we measure characteristics of formal methods artefacts and activities? [4]
    a. How can we measure productivity of proof activities? [1]
    b. How can we measure the size of formal methods artefacts and activities? [1]
3. What are the characteristics of formal methods projects? (including tools and technologies used, industrial domain, and size) [33]

Efficiency is normally measured in terms of output generated per unit of input. In traditional software engineering this could be source lines of code per unit of effort. In formal methods projects, the unit of input remains unchanged (effort), but there are new kinds of output: new kinds of artefacts, including formal specifications, and proofs.

A number of papers have described metrics for formal specifications, although the difficulty of obtaining good data for the study of metrics has been noted by formal methods researchers [7]. Olszewska and Sere [38] summarize prior work in this area and investigate the use of Halstead-derived measures for Event-B specifications. Bollin and Tabareh [4] describe metrics (e.g. logical complexity) used on a formal specification in the Z notation. Staples et al. [44] use lines of specification (analogous to lines of code), for specifications written in scripts for the Isabelle/HOL logic.

Metrics for other formal methods artefacts are largely unexplored. In Andronick et al. [1] it was noted that lines of proof, like lines of code, present issues for size measurement, and that complexity measurement for proofs is far from simple. (For example, proof commands that invoke automated proof search can hide the underlying structural complexity of the proof.) This paper also considered possible effort drivers that may influence the productivity of proof activities. Expertise, collaboration, and tool support were discussed as influencing proof productivity.

One paper [33] reflects on question 3 above, and considers the integration of formal verification techniques and tools with software engineering modeling and design methods. The authors conclude that there is a need to "advance theories, tools and experiments for both verification and design".

In summary there is very little empirical evidence to date on the properties of formal verification activities and artefacts. Measurement seems to be in its infancy and as a result, our empirically supported understanding of verification projects is slim.

4.1.3 Descriptive-Comparative Questions

Easterbrook et al. [14] say that Descriptive-Comparative questions are of the form "How does X differ from Y?". One might even ask "Does X differ from Y?". Such questions contrast characteristics of things over some dimension of difference. There are many potential characteristics and dimensions of interest. A major one is between project contexts – between the three varieties of projects using formal methods identified above in section 3.3 and versus projects using conventional software engineering. Other characteristics and dimensions of

difference arise from the properties identified by the Descriptive and Classification questions above in section 4.1.2, including industrial domains, tools or technologies used, size or complexity of the specifications and systems, and stages of the project lifecycle.

Descriptive-Comparative questions include:

1. How do the size or complexity of formal methods artefacts compare to each other? [4], [44]
2. How do artefacts and activities (including process lifecycle) in formal methods projects differ from those in conventional software engineering? [1], [19]
3. How does productivity in formal methods projects compare to productivity in conventional software engineering projects? [22],[30], [50]
4. How does the productivity of large formal methods projects or systems compare to small ones? [50]
5. In software engineering using formal methods, how does the productivity of maintenance compare to productivity of original development? [1]
6. How do the productivity of the three GQM project contexts compare? [22]

In our study of the relationships between the size of artefacts in the L4.verified project, Staples et al. [44] showed that, for this project, there were very high correlations between the size of abstract specifications, executable specifications and C code when measured as the number of lines, but only weak correlations when measuring size using Cosmic function points. Complexity was not measured in this study.

Bollin and Tabareh [4] compared a number of specification-based measures and code-based measures. They found strong correlations between line counts, some structure metrics and "conceptual" complexity.

Ghezzi et al. [19] note that "there is still a serious mismatch between verification and modern development processes". Andronick et al. presented a descriptive process model for L4.verified which revealed significant differences in activities and artefacts for this process when compared with conventional software engineering [1].

Formal methods is often thought to be expensive compared to conventional software engineering [22]. However, a survey [50] reported on 62 industrial projects using formal techniques. In their report they provide data and observations concerning cost, productivity and quality in projects from various domains and sizes. In general, opinion on the projects was that, compared with traditional software engineering, 92% showed improved quality, 37% reduced cost, and 35% reduced time. One disappointment was the observation that in the majority of projects surveyed no quantitative data was available on relative time and costs. They state that "many respondents did not know the cost implications of their use of formal methods and verification". The L4.verified project claimed that their overall cost of development was less than would have been expected for conventional development of highly critical software [30]. They state "with the right approach, formally verified software is actually less expensive than traditionally engineered 'high-assurance' software". There is very little research on maintenance in formal methods projects. Even if formal verification removes all defects, so that there are no corrective changes, maintenance may still occur for perfective or adaptive change. Some authors [40] have been skeptical that it is feasible to maintain large formal proofs, but Andronick et al. [1] and Klein et al. [30] report on successful maintenance in the L4.verified project. They note that some code changes result in only local updates to the proof, but that other changes (such as those impacting system invariants) can lead to large amounts of proof rework. The nature of proof maintenance

and proof rework is not yet well understood.

4.2 Base Rate Questions

Base Rate questions attempt to understand the "normal patterns of occurrence" of the elements of interest. These include Frequency and Distribution questions, Descriptive Process questions, and Relationship questions.

4.2.1 Frequency and Distribution Questions

Easterbrook et al. [14] say that Frequency and Distribution questions include "How often does X occur?" and "What is an average amount of X?". We need answers to these types of questions to be able to understand whether an activity under study is abnormal or outside the bounds of experience. These questions provide statistical analyses of properties identified in research on Descriptive and Classification questions (section 4.1.2). Example questions in our context include:

1. What is the relative frequency of occurrence in practice of the three types of formal methods projects (identified in section 3.3)? [12], [50]
2. What is the variation in productivity for formal methods activities?
    a. What is the size or complexity distribution of formal methods artefacts (e.g. formal specifications, formal proofs)? [1], [30]
    b. What is the distribution of effort for proof activities encountered? [1], [30], [32]

We could find no literature specific to question 1 above. However there are reports of industrial experience in a more general sense from two industrial surveys. The survey of Craigen et al. [12] reported on 12 case studies which applied formal techniques in an industrial setting. A recent review [50] of 62 industrial projects using formal methods reported the rate of use of various techniques with specification/modelling being the most common (>55 projects) and proofs being used in < 15 projects.

Regarding the size of verification artefacts, perhaps the most detailed reports are from the L4.verified project [1][30]. In these they show the growth curve over time measured in lines for the seL4 kernel abstract specification, executable specification, and C code implementation, two refinement proofs, and property proofs such as integrity and confidentiality (information flow). They also discuss the effort expended in the phases of the project and comment on productivity as revealed in the effort/size graphs.

Larsen et al. [32] report the cost distribution over lifecycle phases for conventional development project versus a project using formal specification within British Aerospace. Their results showed formal specification required relatively more time in early lifecycle (described as analysis) but less time in design and implementation. Overall there was not a significant difference between effort required for formal and conventional projects.

4.2.2 Descriptive Process Questions

Easterbrook et al. [14] say that Descriptive Process questions include "How does X normally work?", "What is the process by which X happens?", "In what sequence do the events of X occur?", "What are the steps X goes through as it evolves?", and "How does X achieve its purpose?". Descriptive Process questions establish how things are done. In section 3.3 we identified formal methods activities as a unit of analysis. Descriptive Process questions characterize these activities, processes, and process lifecycles:

1. What is the process lifecycle by which proofs are developed? [1]
2. What are the detailed steps involved in the development of individual formal methods artefacts (e.g. formal specifications or formal proofs)? [1]

3. What formal verification activities can be performed concurrently within a team? [11] [49]

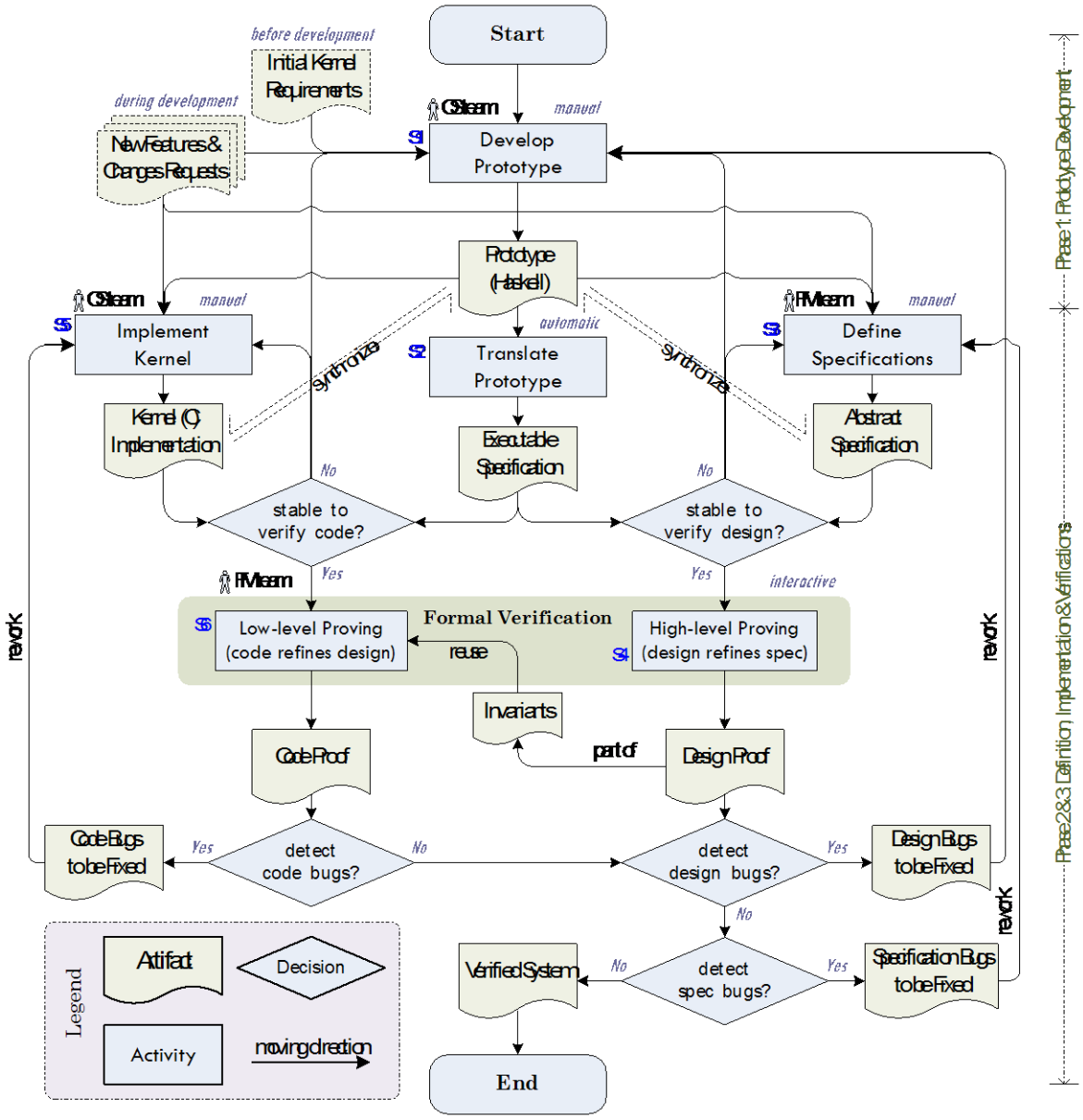

Figure 1 A descriptive process model for a formal verification process, from [51]

Regarding these questions, Figure 1 shows the descriptive process model for the seL4 development process (from [51]). This shows the artefacts, activities and phases of this project. Note that this does not necessarily describe processes for other projects using formal methods. The related paper by Cock et al. [11] (with reference to question 3 regarding concurrent activities) observed, "four people worked on (the) proof concurrently and independently". The FM team managed to design refinement calculi for each of the major proof steps that allowed the work to be divided up among members of the team [11] [49]. To date, however, there is a paucity of literature addressing descriptive processes.

Author's preprint of paper published in Information and Software Technology, Vol. 60, April 2015, pp. 102-112
http://dx.doi.org/10.1016/j.infsof.2014.11.005

4.2.3 Relationship Questions

Easterbrook et al. [14] say that Relationship questions include "Are X and Y related?" and, "Do occurrences of X correlate with the occurrences of Y?". Relationship questions expand on the Descriptive-Comparative questions discussed above in section 4.1.3, usually by statistically investigating correlations.

Example questions include:

1. What are the size relationships between formal methods artefacts (e.g. between formal specifications, designs, and code)? [1], [4], [31], [44]
2. How are project characteristics (e.g. staff experience, tools or technology used, or project size) related to productivity in formal methods projects? [1], [50]
3. Which of the cost drivers for conventional software engineering have the greatest correlation with proof effort in formal methods projects? [1], [5]
4. How is the re-verification effort for maintenance activities related to the effort for initial verification and to the size of the maintenance changes? [30]
5. How are characteristics of formal specifications, properties, or code related to effort in formal proofs? [30]
6. What is the (normalized) cost of formal verification compared with testing? [10], [22], [23], [27], [45], [50]
7. What is the quality of code for which a refinement proof has been completed versus code for which no proof has been completed? [23], [43]

Again, in the L4.verified project there is a strong relationship between the size of a full formal specification of system functionality and the size of the code that implements it [44]. A related piece of research by Lai and Huang [31] reported on a sizing model for "formal communication protocol specification" and its implementation. In this work they investigated the relationship between the size of the informal specification and the formal specification and its implementation. Their results showed strong relationships between Estelle specifications and generated C code. Correlations between specification-based measures and code-based measures were investigated by Bollin and Tabareh [4] using data from a Z specification and ADA implementation. They found high correlation, for example, between "number of physical lines" of code and "conceptual complexity" of the specification.

Bowen and Hinchey [5] discuss COCOMO cost drivers in the context of formal methods noting that factors such as required software reliability, product complexity, and execution time constraints are likely to be influential. They believe other COCOMO factors are not likely to be significant but that new drivers for specification language experience, formal methods experience and domain experience are likely to be important. But again, empirical evidence is lacking.

Estimation for formal methods projects is hard [6] [45], and it is not known how to calibrate measures of formal methods work to traditional metrics approaches [18]. Stidolph and Whitehead [45] also note that estimation techniques are unsatisfactory. In the case of the AAMP-FV project it was found that expertise gained on prior projects significantly improved productivity by almost an order of magnitude [50].

Sobel and Clarkson [43] conducted a classroom experiment in which one team implemented an elevator system using undefined processes (described as the control group), while a second group used various formal specifications. They found that the six defined test cases were satisfied by 100% of the Formal methods group and 45.5% by the "control" group.

In comparing the implications of defect occurrence for software quality, Holzmann [23] notes that conventional testing excels in finding defects which have high probability of occurrence, while verification techniques have an advantage with respect to low probability of occurrence because they look for possible behaviors rather than probable. Holzmann [23] also presents speculative economic models for cost and benefit of various kinds of software verification techniques, including Formal methods.

An industrial survey [50] reported that "three times as many projects reported a reduction in time when using formal techniques rather than an increase. Five times as many reported reduced costs, and 92% reported increased quality".

There has been little analysis of how characteristics of formal specifications and properties to be proved of code affect the cost of formal verification. However, it is clear from the experiences in the L4.verified project that this can be significant – the effort to verify an integrity property for the seL4 abstract specification was many times less than the effort required to verify a confidentiality property for the same specification. [30] But it is unclear how to characterize the causes of this difficulty.

Discussion on formal verification versus testing appears in a number of papers. King et al. [27] observe in a case study that code proofs were more efficient at finding errors than unit testing. Stidolph and Whitehead [45] discuss the difficulties in assigning dollar values to the costs and benefits of Formal methods compared to traditional development, noting cases in the literature of lower productivity, equivalent productivity, and in the case of maintenance, productivity enhancement. Holzmann [23] notes that for "call processing code" software verification techniques "can increase the number of defects intercepted during system testing ten-fold when compared with traditional testing".

Some industrial projects have experienced overall cost reductions when using formal specifications [10] [22] [50]. King et al. [27] report on a study of Z specifications and SPARK code concluding that "the Z proof appears to be substantially more efficient at finding faults than the most efficient testing phase". In a literature study Stidolph and Whitehead [45] report that of the 66 cases studied "only 7 claimed equal or better cost numbers (for formal methods) as compared to traditional development", but that the cost impact of using formal specification varies. However they also note that, in general, "there is little or no history available for the cost and schedule impact of using formal methods".

Klein et al. [30] reported that the cost of re-verification varied significantly, depending on the nature of the change (whether to the top-level specification, system design, or invariant conditions). One change to less than 5% of the code base resulted in proof rework equivalent to 17% of the entire original proof effort [30], but in many cases the effort to re-verify the system was low and proportional to the effort for the change to the code.

Again we note, however, that evidence concerning specification is more available than for verification.

4.3 Knowledge Questions

Easterbrook et al. [14] list three types of Knowledge questions:

- Causality questions, of the form "Does X cause Y?", "Does X prevent Y?", "What causes Y?", "What are all the factors that cause Y?", and "What effect does X have on Y?"
- Causality-Comparative questions, of the form "Does X cause more Y than does Z?" and "Is X better at preventing Y than is Z?"

- Causality-Comparative Interaction: questions, of the form "Does X or Z cause more Y under one condition but not others?"

In the complexity of industrial practice it can be difficult to identify causality because of lack of control over the variables. So although relationships and correlations might be established via Relationship questions, it may not be possible to create controlled experimental environments to understand causality. However, it may be possible to gain understanding of causality by drawing on the experience of verification experts. We do not identify specific Knowledge questions here, but note that any of the example Relationship questions from section 4.2.3 above could be cast as investigations of causality rather than correlation.

4.4 Design Questions

Easterbrook et al. [14] say that Design questions include "What's an effective way to achieve X?" and "What strategies help to achieve X?". In this paper we are primarily concerned with achieving productivity in formal methods projects, and so our Design questions focus on predicting productivity and improving productivity. This may be done by modulating the project context, activities, or artefacts of formal methods projects.

Here we note only a few Design questions – there are far too many possibilities for one paper to cover exhaustively:

1. Can we combine formal verification with conventional testing to improve software quality at reasonable cost? [18]
2. How can formal methods activities be effectively integrated into existing development process models? [16], [26], [35]
3. How can we estimate the cost and effort of formal methods activities and projects? [6], [18], [45]
4. How can we best combine interactive proof and proof automation to achieve high proof productivity during initial proof development and subsequent proof maintenance? [30]
5. How can we best reuse specification and proof to improve formal methods productivity? [1]
6. How should we optimally allocate effort between tool development and proof work in a formal verification project? [1]
7. In which order should work be done in a formal methods project to optimize overall productivity? (bottom-up, top-down, or middle-out?) [1]

Gerhart et al. [18] state that "a research challenge is to better integrate the use of formal methods with other assurance techniques, such as testing and reliability modeling". Formal methods and conventional testing may be complementary [43].

In an early paper, Kemmerer [26] described how formal verification might occur in industry (a) after software development, or (b) in parallel with development but by a separate team, or (c) integrated with the development process. McMillan [35] outlines some principles for the use of formal verification including early lifecycle verification and simplification through decomposition. More recently Stidolph and Whitehead [45] observe that "use of formal methods in industry is still in the investigation stage" but they do not provide any detailed treatment of process integration or estimation methods.

The last four questions are issues for what could be called a new discipline of Proof Engineering. Solutions for these problems are technical challenges for formal methods support systems, including proof automation, interactive theorem provers, theory libraries, specification editors,

and specification animation tools. The L4.verified project has previously identified [1] [30] such issues from their experience in large-scale formal verification, but these challenges are essentially universal for proof in formal methods projects.

5. METRICS AND GUIDANCE ON DATA

Following GQM, the previous sections defined our space of research Goals and research Questions that summarize the numerous research opportunities– this section identifies some of the related Metrics.

SEI's method [39] notes that measurement requires: entities (objects of interest), attributes (characteristics of entities), and rules (for assigning values to attributes). Measures can be defined on various kinds of scale types (nominal, ordinal, interval, ratio, or absolute), and not all of these are quantitative. As our knowledge about an entity increases, we may shift to use measures with increasingly powerful scale types. Bush and Fenton [9] noted that metrics in use could be categorized into those for products (specifications, designs, code, etc.), processes (constructing specification, design, testing, etc.), and resources (personnel, teams, organizations, software, hardware, etc.). It was further noted by Kitchenham et al. [28] that the particular software being studied would influence not only the relevant metrics, but also the relevant relationships between metrics. Context is an important characteristic that needs to be specified and/or measured when considering software engineering metrics [28].

The method used here is to first identify the activities and artefacts in the three project types of section 3.3. Using the research questions derived in Section 4 we identify the metric types that would be needed to answer each question. The result of this analysis is shown in Table 1.

5.1 Analysis of Metrics & Questions by Entity

Table 1 shows that the activities and artefacts identified include a number that would not appear in a traditional software development lifecycle, specifically proof artefacts and activities. The abstract metric types identified are consistent with traditional software engineering, but possibly quite different in their measurement. Thus metrics for size, effort, productivity, cost drivers, complexity and so on are indicated but how these might be measured is a complex issue in some cases and not addressed in research to date. In the following section we provide an example to illustrate this, taken from a small survey of persons in the L4.verified project [30]. We do not delve further into metrics in this paper – these are open research issues in the measurement for formal methods. Thus although the work identifies the need for metrics concerning size, complexity, cost drivers and so on, significant research opportunity exists to explore the definition of these metrics in this context and their efficacy in productivity measurement. It is clear however, that one necessary task in any empirical research in this domain will be the need to clearly and explicitly define the metrics being used and further there will be a need to reach consensus based on evidence of the validity of proposed metrics.

We do not have deep knowledge as to whether the definition of the metrics and collection of data for these metrics will be more difficult than for data about conventional software engineering. Our experience to date is that some proof measures can be complex and require a deep knowledge of proof representation. Careful empirical validation of proof metrics will be required. In any significant formal methods project, formal specifications, designs, and proof artefacts should be version-controlled and managed under fairly standard change control procedures. They will either be managed in a structured representation or be parseable source texts. So, repository mining techniques will provide data for artefact metrics. The problem of collecting data about proof activities should similarly be no harder than collecting data on

conventional software engineering activities. At a coarse level, timesheet data can provide information about effort. Repository mining techniques may again help, to cross-validate and decompose raw timesheet data into metrics on finer-grained activities.

5.2 Possible Size Metrics for Proofs

As an initial exploratory study, we used a brainstorming session for potential size measures followed by a questionnaire asking about "Importance of Proof Measures for Predicting Proof Effort" for each size measure, with responses on a 5-point Likert scale. The scale was defined by points 1 ("Not important"), 3 ("Moderately important"), and 5 ("Very important").

The sample population was a convenience sample: members of the L4.verified project [30]. Seventeen respondents participated and anonymous responses were accepted. Two responses had ambiguous markings and were excluded. Descriptive statistics for the remaining 15 responses are shown in Table 1. For the measures identified in brainstorming as potentially important for predicting proof effort, and thus understanding productivity, we summarized responses about their importance, and whether they are trailing or leading indicators (i.e. might be known before commencing the proof).

Table 1 Possible size metrics for proofs, showing descriptive statistics of frequency of responses on their importance for predicting proof effort

Trail/ Lead	Measure	Min	Max	Mean
T	source lines of proof (i.e. normalized)	2	5	3.6
T	interconnectedness: graph metrics on theorem dependencies, e.g. depth/width/degree distribution	2	5	3.5
L	input artefact size (e.g. spec size etc.)	2	5	3.5
L	need for framework development	1	5	3.3
T	raw lines of proof	2	5	3.1
T	number of stated theorems	2	5	3.1
L	number of hand-written definitions	2	4	3.1
T	theorems' term size or term depth	1	4	3.1
T	ratio of high- to low-level proof methods used	1	4	3.0
T	total number of theorems generated by the theorem prover	1	4	2.8
T	time to machine-check a proofs	1	5	2.6
L	presence of known-"hard"-to-reason-about constants	1	4	2.6
T	size of binary image of theorem prover	1	4	2.1

This data shows divergence of views concerning importance for effort prediction and also shows a wide range of metrics that could potentially be used. The measures identified are largely concerned with aspects of the problem or artefact size or complexity. It is interesting to note the absence of formal methods experience or domain experience measures as suggested as being important by Bowen and Hinchey [5] likely because these were constants in the context of the study. This data reveals an opportunity for empirical researchers to test the efficacy of these

metrics for effort prediction in formal verification projects. As mentioned other factors are also possible. For example, the tools or specification languages used may impact productivity.

6. RELATED WORK

It may be thought that many research questions for conventional software engineering could apply directly to formal methods – after all, from one point of view, formal methods is just another technique to help the development of software solutions to user problems. However, our review of the research question catalogue from SEI IPRC report [17] identified few that are directly relevant to formal methods. Nonetheless, some of their research questions on conventional verification could apply for formal verification. These questions focus on the scale-up of verification approaches, the role of automation in verification, and the relationships between product quality and verification processes.

The industrial adoption of formal methods (or lack of it to date) has been a concern within the formal methods community. As a result there have been a number of papers reporting on myths [22][6], guidelines [5][8][32], and management issues [45] for formal methods projects. There have also been some surveys of industrial use of formal methods [3][50][12], and some surveys of published experience reports [7]. Individually and together, these papers also identify a range of research questions and evidence on productivity for formal methods.

To our knowledge, our paper is the first that provides a structured research agenda for the empirical investigation of productivity in the application of formal methods to software development.

7. FUTURE WORK

The major opportunity for future work arising from this paper is to empirically study the research questions we have identified. Many of these could be investigated in retrospective project analyses provided that version control repositories regularly keep track of changes to artefacts, provided that "bug-tracking" databases are used to record the reasons for repository changes, and provided that base metrics such as effort and size are analysed along with appropriate context and cost driver details. Public repositories such as the Archive of Formal Proofs (http://afp.sourceforge.net/) are ripe for study, although they provide only a very limited range of formal methods and software engineering projects.

In this paper we have only considered one GQM focus area (productivity). Future work could consider others. For example, there has been prior work on formal methods with respect to usability [13], readability (and understandability) [34], maintainability [1], and reuse [8] [24].

At an extreme formal methods offers the possibility that software is bug-free, which radically changes the nature of cost-effectiveness and leads to a broad new area of empirical software engineering research. Perhaps an important observation for empiricists was made by Jackson [25] who argued that the processes of conceiving and designing large artefacts needs intuition and informal processes. Desired properties and behaviours must be understood before they can be subjected to formal verification. This interplay between formal and informal processes may itself form a part of the future empirical enquiry. It is our belief that future empirical research in formal methods will need to be driven by industrial need. It is important that the research occurs in the context of formal methods projects and the focus of the empirical research should be driven by the needs of those projects and the context in which they occur. Thus, similar to the development of functional size metrics in commercial software development, it is expected that metric definition and standardization for this setting should be carried out in association with industry, being cognisant of their needs. Additionally any future research roadmap in the field of

proof engineering should also be developed in partnership with industry and focussed on their needs.

8. CONCLUSIONS

Formal methods has long held the promise to radically improve software engineering, by qualitatively changing the nature of software quality and quality assurance. The use of formal methods is becoming feasible in practice, but there are barriers to its wider adoption. A major barrier is the lack of understanding about the cost-effectiveness of formal methods, and in particular a lack of understanding about cost, estimation, and productivity.

This paper has set out an empirical research agenda for the study of productivity in software engineering projects using formal methods. We used the GQM approach as an investigative framework [2][37], the SEI measurement method to cover a range of research and measurement goals [39], and Easterbrook et al.'s research question categories to derive a set research questions [14]. Face validity of the research questions was given by reference to the existing literature. Using the research questions, we mapped process-identified artefacts and activities to derived abstract metrics. Although detailed metrics for processes and artefacts could not be derived in this paper, we provided a space of potential metrics for proof effort estimation using a small survey-based approach. This resulted in 13 possible metrics about proofs, and we have noted that more proof size measures could be identified. Metrics identified in this way could be used in the future in empirical studies of formal methods projects.

There is also significant opportunity to develop more questions, refine and validate the metrics identified in this paper, and importantly, to conduct empirical studies to expand our knowledge of formal methods productivity.

The research opportunities and potential benefits to practice are highly significant. Formal specifications and formal proofs are similar but different to normal software engineering artefacts. The similarities provide an opportunity to leverage empirical software engineering approaches on this new problem domain. We believe that the differences are significant enough to warrant the formation of a new specialist discipline of 'proof engineering'.

9. ACKNOWLEDGMENTS

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

10. REFERENCES

[1] 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 *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*, IEEE, pp. 1002-1011.

[2] Basili, V., Caldiera, G., and Rombach, D. H. (1994) The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*, Wiley.

[3] Bloomfield, R. E., and Craigen, D. (2000) *Formal methods Diffusion: Past Lessons and Future Prospects*, Adelard.

[4] Bollin, A., and Tabareh, A. (2012) Predictive Software Measures based on Z Specifications - A Case Study. *Proceedings of the 2nd Workshop on Formal methods in the Development of Software*, pp. 33-40.

[5] Bowen, J. P. and Hinchey, M. G. (1994) Ten commandments of formal methods. In *IEEE Computer*, vol. 28, pp. 56-63.

[6] Bowen, J. P. and Hinchey, M. G. (1995) Seven more myths of formal methods. *IEEE Software*, vol. 12, pp. 34-41, Jul. 1995.

[7] Bowen, J. P. and Hinchey, M. G. (1997) The Use of Industrial-Strength Formal methods, *Proc. COMPSAC'97*, IEEE Computer Society

[8] Bowen, J. P. and Hinchey, M. G. (2012) Ten commandments of formal methods...Ten years later, *IEEE Computer*, pp.40-48.

[9] Bush, M. E. and Fenton, N. E., (1990) Software measurement: a conceptual framework, *Journal of Systems and Software*, vol. 12, pp. 223-231.

[10] Clarke, E. M., and Wing, J. M. (1996) Formal methods: state of the art and future directions. *ACM Computing Surveys*, vol. 28, pp. 626-643.

[11] Cock, D., Klein, G., and Sewell, T. (2008) Secure microkernels, state monads and scalable refinement. In *Proc. 21st TPHOLs*, ser. LNCS, vol. 5170. Springer, pp. 167-182.

[12] Craigen, D., Gerhart, S., and Ralston, T. (1993) *An international survey of industrial applications of formal methods, Vol. 2* Case Studies, U.S. Department of Commerce, Technology Administration, NIST Computer Systems Laboratory, Gaithersburg, MD.

[13] Degani, A., Heymann, M., and Shafto, M. (2013), Modeling and formal analysis of human-machine interaction, in *The Oxford Handbook of Cognitive Engineering*, Oxford University Press, pp. 433-448.

[14] Easterbrook, S., Singer, J., Storey, M., and Damian, D. (2008) Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering*, Ed. by F. Shull, J. Singer, D. Sjoberg, Springer, pp. 285-311.

[15] Fenton, N. E. and Kaposi, A. A. (1989) An engineering theory of structure and measurement. In B.A. Kitchenham and B. Littlewood (eds.) *Software Metrics. Measurement for Software Control and Assurance*, Elsevier, pp.27-62.

[16] Fitzgerald, J. S., Larsen, P. G., and Larsen, P. G. (1995) Formal specification techniques in the commercial development process. In *Position Papers from the Workshop on Formal methods Application in Software Engineering Practic*e, ICSE-17.

[17] Forrester, E. (ed.) (2006) *A Process Research Framework*, SEI.

[18] Gerhart, S., Craigen, D., and Ralston, T. (1993) Observations on industrial practice using formal methods. In *Proc. 15th ICSE*, IEEE, pp. 24-33.

[19] Ghezzi, C., Sharifloo, A. M., and Menghi, C. (2013) Towards agile verification, in *Perspectives on the future of software engineering*, Springer-Verlag, pp. 31-47.

[20] Gorschek, T., Tempero, E., Angelis, L., (2014) On the use of software design models in software development practice: An empirical investigation, The Journal of Systems and Software, September, Vol. 95, pp. 176-193.

[21] Gregor, S. (2006) The nature of theory in information systems. *MIS Quarterly*, vol. 30, no. 3, pp. 611-642.

[22] Hall, A. (1990) Seven myths of formal methods. *IEEE Software*, vol. 7, pp. 11-19.

[23] Holzmann, G. J. (2001) Economics of Software Verification. In *PASTE'01*, ACM.

[24] Iliasov, A., Troubitsyna, E., laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., and Latvala, T. (2010) Supporting reuse in Event B development: Modularisation approach. In *Abstract state machines, Alloy, B and Z*, LNCS, 5977, pp.174-188.

[25] Jackson, D., Thomas, M., and Millett, L. I. (editors) (2007) *Software for Dependable Systems: Sufficient Evidence?* The National Academies Press.

[26] Kemmerer, R. A. (1990) Integrating formal methods into the development process. In: *IEEE Software*, vol. 7, Sep., pp. 37–50.

[27] King S., Hammond, J., Chapman, R., and Pryor, A. (2000) Is Proof More Cost-Effective Than Testing?, *IEEE Trans. on Software Engineering*, vol. 26. no. 8, pp. 675–686.

[28] Kitchenham, B.A ., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El-Emam, K., and Rosenberg, J. (2001) *Preliminary guidelines for empirical research in software engineering*, National Research Council of Canada, January, 27 pages.

[29] Klein, G. (2009) Operating Systems Verification – An Overview, Sadhana Academy Proceedings in Engineering Sciences, Vol.34, no. 1. pp. 27-69.

[30] Klein, G., Andronick, J., Elphinstone, K., Murray, T., Sewell, T., Kolanski, R., and Heiser, G., (2014) Comprehensive Formal Verification of an OS Microkernel. In *ACM Trans on Comp. Sys.*, vol. 32, no. 1. pp. 2:1-2:70.

[31] Lai, R. and Huang, S. J. (2003) A model for estimating the size of a formal communication protocol specification and its implementation. In *IEEE Transactions on Software Engineering*, vol. 29, no. 1, pp. 46-62.

[32] Larsen, P. G., Fitzgerald, J., and Brookes, T. (1996) Applying formal specification in industry. *IEEE Software*, vol. 13, May, pp. 48–56.

[33] Liu, Z. and Venkatesh, R. (2008) Methods and tools for formal software engineering. In *Verified Software: Theories, Tools and Experiments*, ed. by B. Meyer and J. Woodcock, LNCS 4171, pp. 31-41.

[34] Mashkoor, A. and Jacquot, J., (2011) Stepwise validation of formal specifications, *Proc. 18th Asia Pacific Software Engineering Conference*, pp.57-64.

[35] McMillan, K. L. (1994) Fitting formal methods into the design cycle. In *31st Design Automation Conference*, ACM, pp. 314–319.

[36] Meltzoff, J. (1998) *Critical Thinking about Research: Psychology and Related Fields*. Washington D.C. American Psychological Association.

[37] Mendonca, M. G., Basili, V. R., Bhandari, I. S., and Dawson, J. (1998) An approach to improving existing measurement frameworks. In *IBM Systems Journal*, vol. 37, no. 4.

[38] Olszewska, M. and Sere, K. (2010) Specification Metrics for Event-B Developments. In: *CONQUEST 2010*, 20-22 September 2010, Dresden, Germany.

[39] Park, R. E., Goethert, W. B., and Florac, W. A. (1996) *Goal-Driven Software Measurement – A Guidebook*. Technical Report CMU/SE-96-HB-002, Software Engineering Institute.

[40] *Peer Review of a Formal Verification/Design Proof Methodology*. NASA Conference Publication 2377, Jul. 1983.

[41] Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008) Systematic mapping studies in software engineering. In *Proc. 12th International Conference on Evaluation and Assessment in Software Engineering*, BCS, pp. 68-77.

[42] Petre, M., (2013) UML in practice, In Proc. of the 2013 International Conference on Software Engineering, ICSE '13, IEEE Press, Piscataway, NJ, USA, pp. 722-731.

[43] Sobel, A. E. K. and Clarkson, M.R. (2002) Formal methods application: An empirical tale of software development, *IEEE Transactions on Software Engineering*, vol. 28, no. 3, March, pp. 308-320.

[44] Staples, M., Kolanski, R., Klein, G., Lewis, C., Andronick, J., Murray, T., Jeffery, R., and Bass, L. (2013) Formal Specifications Better than Function Points for Code Sizing. In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, IEEE, pp. 1257-1260.

[45] Stidolph, D., and Whitehead, J. (2003) Managerial issues for the consideration and use of formal methods. In *Proc. FME 2003*, Springer, LNCS 2805, pp. 170–186.

[46] Van Lamsweerde, A. (2000) Formal specification: a roadmap. In *Proc. of the conference on the future of software engineering, ICSE 2000*, ACM, pp. 147-159.

[47] Van Solingen, R., (1999) *The goal/question/metric method*, McGraw Hill Publishing Company, England, 195pp.

[48] Vinter, R., Loomes, M., and Kornbrot, D. (1998) Applying software metrics to formal specifications: A cognitive approach. In *Proc. 5th International Software Metrics Symposium*, IEEE Computer Society, pp. 216-223.

[49] Winwood, S., Klein, G., Sewell, T., Andronick, J., Cock, D., and Norrish, M. (2009) Mind the gap: A verification framework for low-level C. In *Proc. 22nd TPHOLs*, ser. LNCS, vol. 5674. Springer, pp. 500–515.

[50] Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. (2009) Formal methods: Practice and experience. *ACM Computing Surveys*, vol. 41, pp. 19:1–19:36.

[51] Zhang, H., Klein, G., Staples, M., Andronick, J., Zhu, L., and Kolanski, R. (2012) Simulation Modeling of a Large-Scale Formal Verification Process. In *Proc. of the Int. Conf. on Software and Systems Process (ICSSP 2012)*, IEEE, pp. 3-12